Integrated Scheduling of Hard Real-time and Multimedia Tasks

A. Samuel Thinagar and Lawrence Jenkins
Dept. of Electrical Engineering,
Indian Institute of Science
Bangalore 560012
080-293-2692
lawrn@ee.iisc.ernet.in

Abstract - In this paper, we develop a tool for carrying out acceptance tests in a multiprocessor system, for workloads that include both sporadic hard tasks with specified resource requirements, and multimedia tasks with firm (m, k) guarantees. The tool is a two-level scheduler. The higher-level server planner generates server arrangements for the multimedia server, such that it can meet the (m, k) firm guarantees, and can execute optional task instances as well, while the lower-level feasibility checker determines the schedulability of the sporadic tasks under the specified server arrangement. The two levels iterate until schedulability is determined. The tool has been validated through simulation studies.

1. INTRODUCTION

Applications such as the control systems of fly-by-wire aircraft and autonomous vehicle controllers are examples of real time systems in which one needs to process the control information as well as to provide support for audio and video processing. Unlike embedded real-time applications such as automobile cruise control and process control, where the workload is known, and can be scheduled off-line [1], the applications considered in this paper have dynamic workloads. with a mixture of hard and firm constraints. The scheduling environment consists of a systems processor and a set of applications processors. Since the task mix cannot be predicted, task allocation and scheduling must be done on-line. Each time a new task with firm constraints arrives, an acceptance test has to be carried out, to determine whether its real-time performance requirements can be met. If so, the task is allocated to a applications processor, which has been identified as having sufficient resources to schedule the new task without compromising the minimum performance guarantees that have been provided to the tasks which are already present.

At the applications processor, a local scheduler dispatches the instances of the new task, along with those of the rest of the tasks that are already present, through a specified priority scheduling scheme such as Earliest Deadline First (EDF). Although some instances may miss their deadlines, the local scheduler follows a task-skipping scheme [2] to ensure that the firm guarantees of the new task will be met.

The feasibility test on incoming tasks is done by a scheduler that runs on the systems processor, along with the other systems tasks of the processor. The work in this paper relates to the design of the scheduler.

2. THE WORKLOAD

The workload consists of a mixture of hard and firm real-time tasks. The hard real-time tasks are sporadic. Each of them is characterized by an arrival time, a ready time, a worst-case execution time and a deadline. For its execution, a hard task may require resources, such as data structures and communication buffers. Its access to such a resource is pre-specified to be either exclusive, in which no other task can use the resource, or shared, in which multiple tasks all require the resource, and it may be simultaneously allocated to all of them. A hard task cannot be pre-empted; once it begins executing, it runs to completion. We denote by R_h the total utilization of the hard sporadic tasks, where the utilization of each task is defined as the ratio of its maximum execution time to its deadline, relative to the specified time instant, while the total utilization is the sum of the utilizations of the hard tasks that are currently present in the system.

The remaining tasks are multimedia tasks. The multimedia tasks are mission-critical; hence, although they do not have hard deadlines, they cannot be scheduled on a best-effort basis, using the time that is left over from the execution of the hard real-time tasks, as is done is less stringent applications [3]. Instead, these tasks are assumed to satisfy the (m, k)-firm model, as proposed by Hamdaoui and Ramanathan [2] for dealing with periodic multimedia tasks. In this model, one is required to schedule a multimedia task such that at least m instances out of every window of k successive instances of the task meet their deadlines. It is assumed that such scheduling will satisfy the performance requirements; if less than m instances complete, then the multimedia quality is too poor to be acceptable.

Hence, unless one can provide a guarantee of such k out of m completions, the task cannot be accepted.

Multimedia tasks are executed by multimedia servers. Since the scheduling overhead increases with the number of servers, it is preferable to use a single server to handle all multimedia tasks. In this paper, we assume a single multimedia server, although our results extend directly to the multiple server case. The server itself is a periodic task, which is created and scheduled along with the hard sporadic tasks. The maximum computation requirement of the server, R_{s1}, is obtained by adding the utilizations of the individual multimedia streams, where the utilization of a stream is its ratio of its task execution time to its task period. This maximum computation overhead is computed based on the assumption that no task instances are skipped. The minimum requirement, R_{s2}, is similarly obtained, except that the utilization of each process is multiplied by the appropriate m/k ratio, since up to (k-m) instances out of every k successive instances of the task can be skipped.

Each time a new multimedia task is submitted, the server overhead is increased, and the hard real-time tasks have to be rescheduled along with the server. If a feasible schedule cannot be produced, then the new multimedia task is not admitted. Similarly, when a new hard real-time task is submitted, a feasibility test must be carried out. The scheduler has the flexibility of reducing the server overhead to any value not less than R_{s2}, in order to accommodate the newly arrived hard real-time task.

Task scheduling is carried out through a two-level scheme. At task submission time, the lower level scheduler attempts to find a feasible schedule for the hard real-time tasks, while providing server utilization equal to R_{s1}. If the attempt fails, then the higher-level scheduler, known as the server planner, is invoked. The server planner successively reduces the server utilization level and invokes the lower level scheduler to find a feasible schedule. The process terminates when any one of the following three events occurs: (i) a feasible schedule is found (ii) the server utilization reaches R_{s2} without feasibility (iii) the scheduling time is exceeded. In cases (ii) and (iii), the new task is deemed to have failed the acceptance test.

3. TASK FEASIBILITY

Since the feasibility checking problem is NP-complete, we have adopted a heuristic approach to feasibility testing. Each task in the task set has a pre-specified execution time, a deadline, a maximum execution and resource requirements, where resources can be required in either the exclusive or the shared mode. Using these specifications, we attempt to find a feasible schedule through a modification of the technique proposed by Ramamritham, Stankovic and Shiah [4].

The search space of the problem is visualized as an abstract search tree. Initially, the search tree consists of only the root node, which represents a degenerate partial schedule, in which none of the tasks are scheduled. By scheduling any single task, one creates a child of the root node; this child represents a partial solution. The choice of which child to create is made through a heuristic. Similarly, a new node is created as a child of this child node, by scheduling a second task, using the resource left unutilized by the first task. Such creation of child nodes is continued at successive nodes; the objective is to reach a leaf node, which corresponds to a feasible schedule. All intermediate nodes along the path to the leaf node represent partial solutions. A heuristic is employed to choose the child node to explore at each stage. Before making such a choice, one carries out a feasibility check, which involves the concept of strong feasibility.

A partial solution is said to be strongly feasible if every schedule that is obtained by extending the partial schedule with one of the unscheduled tasks is a feasible schedule. Whenever the partial solution corresponding to a node is found to be not strongly feasible, one can conclude that none of the leaf nodes among its descendants can correspond to a feasible solution. Hence, one performs backtracking, to the nearest ancestor for which it is possible to create a new child node. Such backtracking, followed by the expansion of the tree through creating child nodes is continued until either a feasible solution is found, or the allocated time is exhausted.

The search heuristics used for expansion of the tree are earliest deadline first and least laxity first [3]. The schedule is expanded by considering all eligible tasks at the first time instant for which the processor is available. (It has not been allocated to the previously scheduled tasks). After its release, each task is eligible for scheduling in all intervals where its resources are available. We keep track of resource availability through the Earliest Available Time (EAT) vectors, whose n elements correspond to the n resources. The vectors EAT^s and EAT^e correspond to shared and exclusive use, respectively. If $EAT^s(i)$ has a value of t_1, the implication is that resource i has not yet been allocated in the exclusive mode after time t_1, although one or more processes may have been allocated its use in the shared mode after t_1. If $EAT^e(i)$ has a value of t_2, then resource i has not been allocated for use in either the shared or the exclusive mode after time t_2.

Each time a partial schedule is to be expanded, by allocating a processor at time t, the candidate processes are identified through their resource requirements; if a process requires resource i_1 in the shared mode and resource i_2 in the exclusive mode, then the current values of $EAT^s(i_1)$ and $EAT^e(i_2)$ must not exceed t. After the process is scheduled to begin execution at time t, the values of $EAT^e(i_1)$ and both $EAT^e(i_2)$ and $EAT^s(i_2)$ must be incremented to (t+e), where e is the execution time of the process.

Whenever a partial schedule is found to be not strongly feasible, and backtracking occurs, the EAT^c and EAT^s vectors associated with the discarded partial solution are no longer valid, and one must restore the vectors that correspond to the partial solution to which backtracking occurs. This is easily implemented by storing the vectors in an LIFO stack. In this implementation, the element at the top of the stack corresponds to the parent of the current partial solution. If this solution is found to be not strongly feasible, the stack is popped, thus returning to the parent node. If the parent has any unexplored child nodes, then one of these is selected for exploration, and the vectors of the parent are pushed onto the stack. Otherwise, the vectors of the parent are discarded, since it cannot provide a feasible schedule, and backtracking continues, by popping the stack once again.

Selecting a child node corresponds to the scheduling of exactly one process that has not been scheduled in the parent node. In making this selection, one must ignore the following processes: (i) those that have been scheduled in the parent (ii) those that correspond to child nodes that have already been explored. The list of eligible processes associated with a node is kept in the corresponding LIFO stack entry, along with the EAT^c and EAT^s vectors. For each child node, this list is initially created from that of its parent node, by deleting the process that corresponds to the child node. The same deletion takes place in the list of the parent, before it is pushed onto the stack. This ensures that after backtracking takes place, the search will not return to a child that was explored previously. During backtracking, if a popped entry has an empty list, then no child remains to be explored at the corresponding node, so the entry is discarded, and the stack is popped once again.

Since feasibility checking is frequently performed, we taken the following steps to reduce its overhead: i) termination occurs if the number of backtracks reaches a pre-specified value ii) in extending a partial schedule, the number of candidate processes considered by the heuristic is restricted, thus effectively reducing the maximum degree of the interior nodes of the search tree [4]. While these restrictions do increase the probability of the algorithm not finding an available feasible solution, their adoption is an acceptable tradeoff to reduce the computation overhead.

4. MULTIMEDIA TASK SKIPPING

While the feasibility of all sporadic hard tasks must be guaranteed, some multimedia instances can be skipped, due to the (m, k) model. If we consider the special case of scheduling all multimedia task instances, then the feasibility testing approach for sporadic hard tasks directly extends to the integrated workload, but this action increases the probability that some multimedia tasks will fail the acceptance test.

However, for each multimedia stream, we can select some tasks to be skipped, while the others must be executed; the former are labeled as Blue, while the latter are Red. Many combinations of such Blue and Red task labels can be chosen for a stream, such that the m-out-of-k condition is satisfied. It is sufficient if feasibility is satisfied for the integrated workload corresponding to any one acceptable Red-Blue labeling choice for each multimedia stream.

Since it is not possible to consider the feasibility of the huge number of possible choices, we instead use a hierarchical scheduler. The upper level server planner first chooses the overhead allocated to server tasks. Given this overhead, the feasibility of the integrated workload is checked. If feasibility cannot be established, then one returns to the planner, to set a different server overhead; in this way, one iteratively determines feasibility.

5. THE HIERARCHICAL SCHEDULER

The hierarchical scheduler is invoked each time a new sporadic task or multimedia stream is submitted to the system. The server planner first computes the maximum and minimum multimedia server requirements R_{s1} and R_{s2}, respectively, which correspond to no skipping and maximum skipping. We have proved [5] that a necessary condition for schedulability of an integrated workload is that the sum of the utilizations of the sporadic hard tasks and the multimedia tasks does not exceed the number of processors. Hence, if N is the number of processors, we first check whether N is at least equal to $(R_h + R_{s2})$. If this test fails, we need not proceed further, since no feasible schedule exists, and the new task fails the acceptance test.

Otherwise, the server requirement is set at R_{s1}, which corresponds to no skipping of multimedia task instances, and the corresponding value of $(R_h + R_{s1})$ is compared with N. If the test fails, the server planner replaces R_{s1} by $(R_{s1} + R_{s2})/2$, and carries out the test once again. This reduction in R_{s1} continues, until eventually the chosen value of R_{s1} satisfies the test.

Since the test corresponds to a necessary, but not sufficient, condition for schedulability, one still needs to see whether a feasible schedule exists for the specified value of R_{s1}. Hence, the server planner invokes a lower-level feasibility checker, which attempts to generate a feasible schedule. The feasibility checker uses a set of server arrangement plans, which the server planner provides to it, based on the corresponding value of R_{s1}. Each server arrangement plan is given by a (p_s, c_s) pair, which specifies that the server operates with a time period of p_s, and is replenished with service time c_s, such that the ratio c_s/p_s is equal to R_s. The advantage of the server arrangement plan approach is that it abstracts the multimedia task instances, so that the schedulability now involves only the sporadic tasks and a single server, rather than considering the successive instances in each multimedia stream individually. The server planner ranks the server arrangement plans through a heuristic that maximizes the laxity of the sporadic hard tasks. This ordering reduces the average overhead incurred by the feasibility checker.

If none of the server arrangement plans with the specified value of R_{s1} facilitate the meeting of the deadlines of the sporadic hard tasks, this implies that the current choice of R_{s1} consumes too much overhead to permit their feasible scheduling. Hence, the server planner further reduces R_{s1}, by replacing it with $(R_{s1} + R_{s2})/2$, and repeats the exercise with the new value of R_{s1}. On the other hand, if a feasible schedule is found, the planner accepts the value of R_{s1} as the specified server arrangement plant, and accepts the corresponding (p_s, c_s) value as the multimedia server specification.

6. Scheduling of Multimedia Instances

The server planner always chooses a server requirement that allows one to meet the (m, k) firm guarantees of all multimedia tasks. Hence, all task instances that are given the label Red will complete, while those that are labeled as Blue may perhaps get server time.

While there are many valid labeling schemes for a stream with (m, k) firm guarantees, we have chosen one in which the first m instances of the stream are labeled Red, and the next (k-m) instances are labeled Blue. Subsequently, each instance is labeled as follows: the label is Blue if m-1 of the last k-1 instances were labeled Red; otherwise, the label is Red.

To schedule the task instances, the multimedia server uses two queues. All Red task instances are placed in the Red queue, which is scheduled through EDF. This queue has priority over the Blue queue, in which all the Blue task instances are placed. The scheduling algorithm for the Blue queue is FCFS. The completion times of the Red task instances depend on their arrival instants within the individual streams relative to the replenishment times of the multimedia server. Hence, although we can guarantee that all of them will be executed, some of them may miss their deadlines. This is acceptable, given that these deadlines are soft.

Blue tasks execute whenever the server has budget and the Red task queue is empty. A Blue task instance is removed from the queue whenever a subsequent Red instance from the same stream completes its execution, since the Blue task completion no longer has any value associated with it. The percentage of Blue task instances that complete their execution depends on the difference between the c_s/p_s ratio of the server and the overhead associated with the (m, k) firm guarantees of the multimedia streams.

7. Simulation Studies

The direct method of evaluating the hierarchical scheduler is to randomly generate workloads, and then compute the success ratio of the scheduler in determining their feasibility. Here, we need to distinguish workloads that are infeasible from those for which the scheduler fails to find a feasible schedule, although at least one such feasible schedule exists. The former failures should not be considered in the evaluation of the scheduler, since in such cases no scheduler can succeed. The difficulty in using this approach to scheduler evaluation is that scheduling is heuristic, and so there is no way to determine whether a feasible schedule exists, unless the scheduler happens to find one.

As an alternative, we follow the approach from [4], in which feasible workloads are experimentally constructed. Beginning with an empty workload, successive tasks with randomly chosen arrival times, execution times, deadlines and resource requirements are randomly generated and then randomly assigned to individual processors, on which they are scheduled for execution. As such task are added to the workload, the utilization of the processors increases. If any task misses its deadline when scheduled on the assigned processor, it is discarded. After a pre-specified number of such tasks have been discarded, the constructed workload is assumed to have sufficiently high utilization to adequately test the hierarchical scheduler, and it is added to the set of test workloads. We thus get workloads that are known to be feasible, but which use most of the available computational resources. If the hierarchical scheduler has a high success rate in finding such challenging workloads to be schedulable, then we can be confident of its effectiveness in schedulability testing.

In our simulations, we took average values from three simulation runs, for each of which we generated 200 task sets. The system under consideration has 3 processors, and 12 non-processor resources. Each task was assumed to have 0.7 probability of requiring each resource; a random number generator was used to determine resource requirements. For each such requirement of a resource by a task, the usage was randomly chosen as shared, with probability 0.5;otherwise, it was taken to be exclusive. The schedule length was taken to be 300 ms. Tasks were assumed to have execution times which lie in the range [10 ms, 30 ms] with a uniform probability over that range. The relative deadlines were taken from a uniform distribution in the range [60 ms, 90 ms]; each task set had between 40 and 50 tasks.

Three multimedia streams were included in each workload. The baseline stream was taken to have a period of 33.3ms, a computation time of 5 ms, and a (3, 5) firm guarantee. The second stream had a period and an execution time equal to 1.4 times that of the baseline stream, so that the utilization was equal to that of the baseline; the (m, k) value was (2, 3). For the third stream, the corresponding values were 1.7 times and (3, 4), respectively.

Simulation studies showed that the heuristic scheduler achieved a high success ratio in determining the feasibility of these heavy workloads, for which one knows that a feasible schedule exists. This justifies a high confidence level in the effectiveness of the hierarchical scheduler as a tool for the schedulability determination of a workload with both hard sporadic and firm multimedia real-time tasks.

8. CONCLUSION

We have developed a heuristic hierarchical scheduler for multiprocessor systems with random mixed workloads that include both sporadic hard tasks and multimedia tasks with (m, k) firm guarantees. The high-level server planner chooses server plans for determining the server arrangements for the multimedia server, and the low-level feasibility checker determines whether the sporadic hard tasks meet their deadlines under the specified server arrangements. The two levels iterate until the scheduler either provides a guarantee that the deadlines are met, or the new task is deemed to have failed the acceptance test. The hierarchical scheduler has been tested with a set of synthetically generated workloads that are known to be feasible, and has achieved a high success rate in scheduling these workloads.

REFERENCES

1. H. Hansson, H. Lawson, O. Brindal, C. Eriksson, S. Larsson, H. Lon and M. Stromberg, "BASEMENT: An architecture and methodology for distributed automotive real-time systems", *IEEE* Transactions *on Computers*, Vol. 46, No. 9, September 1997, pp. 1016-1027.
2. M. Hamdaoui and P. Ramanathan, "Dynamic priority assignment technique for streams with (m, k) firm deadlines", *IEEE Trans. On Computers*, Vol 44, No. 12, December 1995.
3. Jane W. S. Liu, *Real Time Systems*, Pearson Education (Singapore), 2001.
4. Krithi Ramamritham, John A. Stankovic and Perng-Fei Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, April 1990, 184 - 194.
5. A. Samuel Thinagar, "Overloaded systems: Skip over algorithms and the complexity of (m, k)-firm task scheduling", Master of Engineering thesis, Department of Electrical Engineering, Indian Institute of Science, January 2001.