DISTRIBUTED SCHEDULING
USING BIDDING AND FOCUSED ADDRESSING

Wei Zhao
Krithivasan Ramamritham
Department of Computer and Information Science
University of Massachusetts
Amherst MA 01003

ABSTRACT

In the design of real-time computer systems, the scheduling problem is considered to be an important one and has been addressed by many researchers. However, most approaches have not dealt with task's resource requirements. In this paper, we describe a heuristic algorithm to schedule hard real-time tasks, i.e. tasks that have deadlines, in a distributed system. Salient features of our algorithm are that it takes tasks' resource requirements into account, is dynamic, and is distributed. When a task arrives at a node, the scheduler component local to that node attempts to schedule the task on that node. If the attempt fails, the scheduling components on individual nodes cooperate to determine which node has sufficient resource surplus to finish the task before its deadline. This cooperation occurs through the exchange of state information among nodes. Determination of a good node to send a task to is based on a technique that combines bidding and focused addressing. In the former, a good node is selected based on the bids that nodes send for the task; in the latter, a node that is estimated to have more than sufficient surplus to guarantee the task is said to be good. Thus, focused addressing avoids the communication overheads inherent in bidding. We report on simulation studies performed on the algorithm.

1. INTRODUCTION

Loosely coupled, real-time distributed systems are becoming more prevalent in applications such as nuclear power plants and process control [SCHO84]. These systems contain many tasks which have severe real-time constraints. In fact, these applications require that these tasks have execution deadlines that must be met and are thus said to have hard real-time constraints. In practice, many solutions used today assume complete and prior knowledge of the task sets, including their deadlines, worst case computation times, precedence relations, and resource requirements. Our goal is to develop a flexible scheduling algorithm for loosely coupled distributed systems that does not require complete and prior knowledge, is not computer bound, and quickly adapts to the dynamics of the "entire" distributed system. Our basic algorithm and approach have been reported in [RAMA84]. There we did not take into account tasks' resource requirements. This paper reports on a sophisticated extension to our basic algorithm that does take into account resource requirements of tasks.

2. RELATED WORK

The work described in this paper is not only an extension to our own work, but also to work in the field in general. Most related work on scheduling tasks with hard real-time constraints is restricted to uniprocessor and multiprocessor systems [MUNT70, LIU73, DERT74, JOHN74, GARE75, MOK78, and TEIX78] and typically do not take tasks' resource requirements into account. However, in the work by Leinbaugh [LEIN80 and LEIN82] resource requirements are dealt with. There he developed analysis algorithms which, when given the resource requirements of each task, determine an upper bound on the response time of each task. In his model, a task can be divided into multiple segments and the segments of a task can be executed concurrently on different nodes. While his approach is useful at system design time to statically determine the upper bounds on response times, there is no attempt at dynamically guaranteeing that a new task will meet its deadline.

In other work, Efe [EFE82] and Ma [MA84] use heuristic approaches for related scheduling problems. According to Lenat [LEN82 and LEN83], heuristics are informal, judgmental rules of thumb which come in two
types: those that actively guide the system toward plausible paths to follow, and those that guide the system away from implausible ones. Both Ma and Efe use heuristics to guide their systems away from implausible paths. This approach of only using the second type of heuristic is limited because in the worst case, the exponential search problem cannot be avoided [EFES82 and MA84]. In our heuristic algorithm, we use both types of heuristics.

The remainder of this paper is organized as follows: Section 3 defines the system model adopted in this paper. Section 4 provides an overview of our heuristic approach for incorporating resource requirements into dynamic scheduling in hard real-time distributed systems. While Section 5 briefly discusses how to schedule a task on a node, Section 6 discusses distributed scheduling. The simulation results are presented in Section 7 and compared with two baseline algorithms. Section 8 summarizes the work.

3. SYSTEM MODEL

There are n nodes, N₁, N₂, ..., Nₙ, in a loosely coupled distributed system. Let each node contain a set of distinct resources, R₁, R₂, ..., Rₙ. A resource can be shared by tasks. A resource is an abstraction and can include CPU, I/O devices, files, etc. A resource is active if it has processing power, otherwise, it is passive. For example, a CPU or a physical device is active, but a file is passive. Thus, always, a passive resource must be used with some active resource.

A task is a scheduling entity. The following characteristics of a task, T, are assumed known when it arrives:

1. The worst case computation time, C(T);
2. The deadline, D(T), by which the task must complete;
3. The resource requirements of the task. It is assumed that a task needs its resources throughout its execution. A task will request at least one active resource and zero or more passive resources.

4. OVERVIEW OF THE SCHEDULING SCHEME

Each node in the distributed system has a component of the scheduler. Each component contains a local scheduler, a bidder, a dispatcher, and a node surplus manager.

The local scheduler at a node is invoked when a new task arrives at that node. It decides if the new task can be guaranteed at this node. The guarantee means that no matter what happens (except failures) this task will execute by its deadline, and that all previously guaranteed tasks will also still meet their deadlines. If the new task cannot be guaranteed locally, then the new task is either rejected or is handed over to the bidder task.

The bidder on a node is responsible for determining where a task that cannot be locally guaranteed should be sent. It does this through a combination of focused addressing and bidding. In focused addressing, a task will be sent directly to another node based on its (partial) knowledge about the other nodes in the system. In bidding, the node will send out a request-for-bid message to other nodes with sufficient surplus on resources needed for the task. The node with the task will send the task to the node which offers the best bid. In addition to sending its tasks to other nodes, the bidder task makes bids in response to requests from other nodes.

The dispatcher is the system program that actually schedules the guaranteed tasks.

It should be pointed out that when a node bids for a task, it does not reserve CPU time for that task. Reserving CPU time ties up too many resources for too long a time. Consequently, when a task finally arrives at a bidder node, the node will attempt to guarantee it. In case this guarantee fails, the task may be considered for bidding again should the task's deadline permit, or declared as not "guaranteed".

One of the assumptions underlying our scheduling algorithm is that nodes can estimate the resource usage or resource surplus of other nodes. This requires that nodes keep each other informed about their surplus. This can be done by the node surplus manager in the following way.

The node surplus manager on each node periodically calculates the node surplus. The node surplus provides information about the available time on
resources, after taking into account resource utilization of local tasks, i.e. the tasks that directly arrived at a node, in a given (past) time window. This information is used to predict the resource availability for the tasks from the other nodes in the near future. The computed node surplus is sent to a selected subset of nodes in the system. The selection is to be based on the proximity of the nodes, on who sent tasks to this node recently, and on whether the tasks were guaranteed. The size of the subset is proportional to the amount of surplus.

The steps involved in scheduling a newly arrived task are as follows.

1. When a local task, T, arrives at a node N, the local scheduler is invoked to try to guarantee the newly arrived task on the node. If the task can be guaranteed, it will be put into the schedule which contains all the guaranteed tasks on the node.

Details of our local scheduling algorithm designed to consider not only computation time and deadlines of tasks but also their general resource requirements can be found in [ZHAO85]. For the sake of completeness, we give a brief description of the local scheduling algorithm in the next section.

2. When the local scheduler of node N is unable to guarantee the newly arrived task, T, it attempts to find another node through Focused Addressing. This focused node should have sufficient surplus to guarantee the task. If a focused node is found, the task is immediately sent to that node. In addition to sending the task to the focused node, node N sends a request-for-bid message to a subset of the other nodes. The size of the subset depends on the estimated surplus of the focused node and that of the rest of the nodes. The request-for-bid message also contains the identity of the focused node if there is one.

3. When a node receives the request-for-bid message, it calculates a bid, indicating the possibility that the task can be guaranteed on the node, and sends the bid to the focused node if there is one, otherwise, to the original node which issued the request-for-bid.

4. When a task reaches a focused node, it first invokes the local scheduler to try to guarantee the task. If it succeeds, all the bids for the task will be ignored. If it fails, the bids for the task will be compared and the task will be sent to the node responding with the "best bid".

5. In case there is no focused node, the original node will receive the bids for the task and will send the task to the node which offers the best bid.

6. If the focused node cannot guarantee the task and if there is no good bid available for the task, we assume that no node in the network is able to guarantee the task.

This scheme synthesizes the ideas of focused addressing and bidding, attempting to make them work in a flexible way. For example, if a focused node cannot be found, a rather large subset of nodes, perhaps all the nodes in the network will be sent the request-for-bid message. In this case, the scheme degrades to the bidding scheme. On the other hand, if the surplus of the focused node is sufficiently large, the subset of the nodes to which the request-for-bid message is sent can be relatively small, perhaps even empty. In this case, the scheme degrades to focused addressing.

5. OVERVIEW OF SCHEDULING ON A NODE

In this section, first we introduce the strategy for scheduling tasks on a node. Then we briefly describe the guarantee routine which is used both in scheduling tasks and in making a bid. [ZHAO85] provides a detailed description.

5.1 Scheduling Tasks on a Node

Assume that there are n nodes in a given system, numbered from 1 to n. At any given time, node N_i (i = 1 ... n) has guaranteed a set of tasks S_i and has a full feasible schedule for this set of tasks. (A feasible schedule is a list of tasks that have been guaranteed. With respect to a set of tasks, a schedule is full, if it contains all tasks in the set, otherwise it is partial. A schedule (T_1, T_2, ..., T_p, T_{p+1}) is an immediate extension of the schedule (T_1, T_2, ..., T_p).)

1. If a task has sufficient laxity (deadline - computation time), then focused addressing and bidding may be repeated. However, this will increase the scheduling and communication overheads.

---

105
Suppose task $T$ arrives at node $N_i$. Then the following steps are taken in order to guarantee the newly arrived task $T$.

1. The scheduler component in node $N_i$ decides to guarantee the new task $T$, if and only if, a new full feasible schedule exists for tasks in $S_i \cup \{T\}$. This ensures that the tasks of $S_i$ in the original feasible schedule remain guaranteed. Also, it ensures that the new task $T$ will meet its deadline.

2. If $T$ is guaranteed by node $N_i$ (as stated above), the new full feasible schedule containing tasks in $S_i \cup \{T\}$ replaces the original one. This schedule determines the start time of the tasks in node $N_i$, and will not be modified until another new task is guaranteed by node $N_i$.

3. If the new task $T$ cannot be guaranteed by node $N_i$, that is, there is no full feasible schedule for tasks in $S_i \cup \{T\}$, we use the approach based on bidding and focused addressing to determine if another node is in a position to guarantee task $T$. When such a node is found, $T$ is sent to that node. In any case, the current feasible schedule of node $N_i$ remains unchanged.

5.2 The Guarantee Routine

In scheduling tasks on a node (and in making a bid), we often need to determine whether a full feasible schedule exists for a given set of tasks. We have a guarantee routine, called by the local scheduler (and by the bidder), to make this decision.

The problem of finding a full feasible schedule is, in fact, a search problem. The structure of the search space is a search tree. The root of the search tree is the empty schedule. An intermediate vertex of the search tree is a partial schedule. A leaf, a terminal vertex, is a full schedule. Note that not all leaves will correspond to feasible schedules. The goal of the guarantee routine is to search for a leaf that corresponds to a full feasible schedule.

An optimal algorithm, in the worst case, may make an exhaustive search, which is computationally intractable. In order to make the algorithm computationally tractable, even in the worst case, we take a heuristic approach for the guarantee routine. We have developed a heuristic function, $H$, which synthesizes the various factors of real-time scheduling considerations to actively direct the scheduling process to a plausible path. That is, on each level of the search, function $H$ is applied to each of the tasks remaining to be scheduled. The task with the minimum value of function $H$ is selected to extend the current (partial) schedule. If it is found that a wrong task has been selected, the task with the second minimum $H$ value may be selected, but in the course of scheduling a set of tasks, such backtracking is done only for a limited number of times. We also constrain the search space by looking only at strongly feasible paths, thus preventing the scheduler from looking at implausible paths. A schedule is strongly feasible if it is feasible, all of its immediate extensions are feasible, and none of the resource utilization ratios of the tasks, which are in the task set and have not been scheduled, is more than one. A strongly feasible path in the search space is the path along which all the (partial) schedules developed are strongly feasible. As a result of the above directed search, even in the worst case, our scheduling algorithm is not exponential.

Function $H$ is defined as follows:

$$H(T) = W_1 \cdot X_1(T) + W_2 \cdot X_2(T) + W_3 \cdot X_3(T)$$

where $T$ is a task, $W_1, W_2$ and $W_3$ are weights and

1. $X_1$ takes into account the resource requirements of $T$ as well as the resource utilizations.

2. $X_2$ equals the laxity of task $T$, and hence is given by

$$X_2(T) = D(T) - (EST(T) + C(T))$$

where $D(T)$ is the deadline of task $T$, $EST(T)$ is the Estimated Start Time for task $T$ according to the availabilities of resources it will request; and $C(T)$ is the worst-case computation time of $T$.

3. $X_3 = C(T)$.

One of the main reasons for choosing $H$ to be a linear function of $X_1, X_2$, and $X_3$ is to keep the function computationally simple. Note that, if $W_1$ and $W_2$ are zero and $W_3 = 1$, then the task with the least computation time $C$ will be selected first; if $W_2 = 1$ but $W_1$ and $W_3$ are zero, the task with the least laxity will be selected first; if $W_1 = 0$ then constraints imposed by tasks' resource requirements will not be taken into account. Our simulation studies [ZHAO85] indicate that, when the weights are properly selected, scheduling using this linear function compares favorably with scheduling using some optimal algorithm such as one that exhaustively searches for a feasible schedule. We have developed an algorithm to automatically identify the proper weights for a given application domain. Details of this algorithm will be reported in [ZHAO86].
6. ISSUES IN DISTRIBUTED SCHEDULING

Our simulation studies have shown that our local scheduling algorithm performs very close to an optimal algorithm [ZHAO85]. Hence the performance of the system will heavily depend on how distributed scheduling is done. The issues here are: the estimation and proper use of nodes' surplus, heuristics for choosing focused nodes, strategies for making bids, and evaluation of the bids. We will pay attention to these issues in the following sections.

6.1 Node Surplus: Their Generation and Transmission

As mentioned earlier, the purpose of generation and transmission of node surplus from a node is to help other nodes to correctly make the decision about which node a task should be sent to (during focused addressing) and which nodes the request-for-bid messages should be sent to (during bidding). Obviously, it is neither practical nor possible to let nodes have precise state information about other nodes, one reason being the communication delay involved. Our notion of a node's surplus is an approximation of the node's state, in particular, its ability to guarantee tasks from other nodes.

A node's surplus is in reality a vector, with one entry per resource on that node. Each entry indicates the fraction of time, in a (past) window, during which a resource is not used by the local tasks.

Each node periodically calculates its node surplus and sends it to a subset of the remaining nodes. A node sorts other nodes according to the number of tasks received from them that were guaranteed on this node. Then, according to this sorted node list, a node selects a subset of nodes to send its own current node surplus. The subset is chosen such that nodes in the subset will potentially use this information in deciding whether or not to send a task to this node; The nodes, which recently sent tasks to this node, will be most likely selected. Of course, if the network is small, the surplus information can be sent to all the other nodes.

6.2 Focused Addressing and Requesting for Bids

When a task, T, arrives at a node Ni, the local scheduler is invoked to try to schedule the newly arrived task on the node. If it is impossible to schedule the task, T, locally, node Ni's bidder task comes into the picture.

For j = 1, ..., n and j ≠ i, the bidder task on node Nj estimates

\[ ES(T, j) = \text{number of instances of task } T \text{ that node } N_j \text{ can guarantee} \]

This estimation is made according to the node surplus information available on node Nj. Suppose the computation time of task T is 250. Suppose, with respect to the resources needed by task T, Node Nj has a surplus of 400 in the time interval in which task T must run (i.e., 400 is the minimum of the total idle times on the resources needed by T). Then \( ES(T, S) = 400/250 = 1.6 \).

Node Ni sorts other nodes according to their \( ES(T, j) \), in descending order. The first k nodes are selected to participate in focused addressing and bidding. The value of k is decided such that the sum of \( ES(T, j) \) of the k nodes is \( \geq SGS \), the System-wide Guarantee Surplus. This is a tunable parameter of the system. If the first node \( N_i \) among the k nodes has its \( ES(T, f) \) larger than FAS, the Focused Addressing Surplus, another tunable parameter, node \( N_i \) will be the focused node. The task will be immediately sent to that node. The remaining k-1 nodes will be sent request-for-bid messages.

Here, properly tuning parameters SGS and FAS will be critical to system performance. FAS should be large enough so that the chance of a focused node guaranteeing a task will be high. SGS should not be too high so that not too many messages are transmitted in the network. It should not be too low for this may result in too many remote tasks (i.e. tasks that did not arrive locally) not being guaranteed, not because there are no nodes that can guarantee them, but because the nodes that can guarantee them are not sent request-for-bid messages.

6.3 Bidding, Bid Evaluation, and Respond to Task Awarded

When a node receives a request-for-bid, it calculates a bid for the task. The bid is purely a number which indicates the number of instances of the task the bidder node can guarantee. The calculation is done in two steps: First, the node tries to insert into its schedule as many instances of the remote task as it can so that the resource utilization ratios obtained after taking into account tasks which have been guaranteed in the schedule plus the instances of the remote tasks are not more than 1. Let MaxI be the maximum number of
instances of the remote task which can be inserted. Note that MaxI is an upper bound on the bid. We then perform a binary search between 0 and MaxI to determine the number of instances of the remote task T that this node can actually guarantee under the current circumstances. This number, if above a pre-defined threshold, becomes the bid. The bid is sent to the node which was selected for focused addressing if there is one. Otherwise, the bid is sent to the original node which issued the request-for-bid. The inserted instances of the remote task are removed from the schedule on a bidder's node. Hence the schedule on the bidder's node will not be affected by the bid it makes.

When a node receives a bid for a given task, and the bid is higher than a certain threshold, the node will award the task to the bidding node immediately and all other bids for this task, that arrived earlier or may arrive later, will be discarded. If all the bids, that have arrived, for a given task are lower than the threshold, the node will postpone making the awarding decision until the latest time at which the task must be sent out to be guaranteed at a remote node. At that time, the task will be awarded the highest bidder if any, where the bid should be above a threshold. All the bids that arrive later will be discarded.

When the awarded task arrives at the best bidder, the local scheduler on that node will be invoked to see if the task is guaranteeable. Note that the state of the node may change after making a bid. Hence the task may or may not be guaranteed. If it is not, it is rejected.

7. EVALUATION OF THE ALGORITHM THROUGH SIMULATION

In this section, we will show the simulation results from the algorithms discussed in the previous sections. We first introduce our simulation model, then propose two baseline algorithms whose performances serve as upper and lower bounds. Finally the simulation results are presented and discussed.

7.1 Simulation Model

In the simulation, we assume that there are six nodes in the system. On each node, there are five resources, including two active resources and three passive resources.

There is one periodic task per node. It has a period of 2000 time units and a computation time of 400 units. A periodic task needs all the resources.

Both the computation time and laxity of non-periodic tasks are normally distributed. The mean of the computation time and laxity are 400 and 600 time units respectively. The standard deviation of them are 100 and 300 respectively. Each non-periodic task will require at least one of the active resources and zero or more passive resources. A task's resource requirements are chosen randomly.

The nonperiodic tasks arrive as a Poisson process. The arrival rates of nonperiodic tasks on different nodes may be different, resulting in differences in the loads of the nodes. In order to quantify the differences in the loads of nodes, for the sake of simplicity, in the simulation we assume that the nonperiodic task arrival rates of the six nodes forms an equal-rate sequence, and the rate is denoted as the balance rate. That is, if the balance rate is $B$, and at the first node on average $N$ tasks arrive per unit time, then the ith node has an arrival rate of $N\times l$. For example, if $B = 0.5$, and $N = 2$, then nodes 1 through 6 experience an arrival rate of $2, 1, 0.5, 0.25, 0.125, \text{ and } 0.0625$ respectively. It is obvious that when the balance rate is between 0 and 1, the closer to 1 it is, the more balanced the nodes will be. Later, when we present the simulation results, we use the term system nonperiodic task arrival rate, to refer to the sum of the nonperiodic task arrival rates of all the nodes. For the above example, the system nonperiodic task arrival rate is equal to 3.9375. To normalize the arrival rate, in the rest of this section, we define the task arrival rate to be the average number of task arrivals per 400 time, 400 being the mean of the task computation time.

The simulation model assumes that the scheduler tasks such as the bidder and the local scheduler are executed on a co-processor dedicated to scheduling. The time delay involved in guaranteeing a task is taken into account, which is proportional to the square of the size of the task set processed by the guarantee routine, this being the time complexity of guarantee routine. The time delay for transferring a message, such as a request-for-bid or a bid, is 10 time units, and for transferring a task is 10% of the computation time of the task.
7.2 Baseline Algorithms

The performance of our algorithm will be compared with that of two baseline algorithms. In this subsection, we define these baseline algorithms.

The first baseline algorithm is a Non-Cooperative scheduling algorithm (N.C.). It schedules tasks on a node like our algorithm. But whenever a task cannot be guaranteed locally, the task is discarded. No attempt is made to send the task to other nodes by focused addressing and/or bidding. It is obvious that the performance of the non-cooperative algorithm will not be as good as ours. Therefore, we believe, that this baseline will provide a lower bound.

The second baseline algorithm is a Perfect State Information scheduling algorithm (P.S.I.). That is, in this algorithm, we assume that each node has precise information about other nodes. Further, the system overheads to obtain such information is assumed to be zero. Then, the performance of this algorithm should be better than ours, thus serving as an upper bound for a distributing scheduling algorithm.

7.3 Simulation Results

In a real-time system such as the ones we are interested in, a major performance metric is the guarantee ratio. It is defined as

Guarantee Ratio =

\[
\frac{\text{Total Number of Tasks Guaranteed in the System}}{\text{Total Number of Tasks Generated in the System}}
\]

In Figures 1, 2, and 3, we plot the guarantee ratio \( G \) vs. system nonperiodic task arrival rate \( R \) for different values of the balance rate. Each figure depicts the performance of our algorithm, i.e. the combined bidding and focused addressing scheduling algorithm, as well as two baseline algorithms, i.e. the Non-Cooperative scheduling algorithm (N.C.) and the Perfect State Information scheduling algorithm (P.S.I.).

These preliminary simulation results indicate that in most cases the performance of our algorithm is much better than the lower bound offered by the non-cooperative algorithm, and is close to the upper bound achieved by the perfect state information algorithm, irrespective of values of the balance rate. We are continuing our simulation studies on other aspects of the system and these results will be reported in a separate paper.
8. SUMMARY

The problem of determining an optimal schedule is known to be NP-hard [GRAH79] and is hence impractical for real-time task scheduling. The problem is further complicated when, in addition to computation times and deadlines of tasks, their resource requirements should also be taken into account. It is impossible to find an optimal schedule for a dynamic distributed system given the inherent communication delay.

In this paper, we described a heuristic approach to solving this problem. This heuristic algorithm can be used to perform on-line scheduling of tasks, i.e., to schedule tasks that arrive dynamically and can handle non-periodic tasks even in the presence of periodic tasks, i.e., tasks that are known to occur at regular intervals.

We use a heuristic function to determine a schedule for tasks that will execute on a node. The cooperation among the nodes, needed when a node is unable to guarantee a task, occurs through a combination of bidding and focused addressing. Our simulation studies show that despite the communication overheads, it performs favorably with an algorithm that has perfect state information.

Besides the issue of resource requirements, there are many other interesting aspects in the design of a scheduler for a hard real-time system, such as inter-task constraints, network communication protocols, and the architecture of a node. We, along with the other researchers in our group, are currently working on these problems.

9. REFERENCES


