DISTRIBUTED SCHEDULING OF TASKS WITH DEADLINES AND RESOURCE REQUIREMENTS

K. Ramamritham, J. Stankovic and W. Zhao

COINS Technical Report 88-92

October 3, 1988

DISTRIBUTED SCHEDULING OF TASKS WITH DEADLINES AND RESOURCE REQUIREMENTS *

Krithi Ramamritham
John A. Stankovic
Wei Zhao
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

October 3, 1988

*This work is part of the Spring Project at the University of Massachusetts is funded in part by the Office of Naval Research under contract N00014-85-K-0398 and by the National Science Foundation under grant DCR-8500332.

Abstract

In the design of distributed 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 tasks' *timing and resource requirements*. In this paper, we describe a set of heuristic algorithms to schedule tasks that have deadlines and resource requirements, in a distributed system. These algorithms are dynamic and function in a decentralized manner. When a task arrives at a node, the local scheduler at that node attempts to guarantee that the task will complete execution before its deadline, on that node. If the attempt fails, the scheduling components on individual nodes cooperate to determine which other node in the system has sufficient resource surplus to guarantee the task. In this paper, four algorithms for cooperation are evaluated. They differ in the way a node treats a task that can not be guaranteed locally:

- The random scheduling algorithm: The task is sent to a randomly selected node.
- The focussed addressing algorithm: The task is sent to a node that is estimated to have sufficient surplus to complete the task before its deadline.
- The bidding algorithm: The task is sent to a node based on the bids received for the task from nodes in the system.
- The flexible algorithm: The task is sent to a node based on a technique that combines *bidding* and *focussed addressing*.

Simulation studies were performed to compare the performance of these algorithms relative to each other as well as with respect to two baselines. The first baseline is the non-cooperative algorithm where a task that cannot be guaranteed locally is not sent to any other node. The second is an (ideal) algorithm that behaves exactly like the bidding algorithm but incurs no communication overheads. The simulation studies examine how communication delay, task laxity, load differences on the nodes, and task computation times affect the performance of the algorithms. The results show that distributed scheduling is effective even in a hard real-time environment and that the relative performance of these algorithms is a function of the system state.

Notations

1. Task Parameters:
   - T: a task,
   - C(T): the worst-case computation time of task T,
   - D(T): the deadline of task T,
   - K: the ratio of mean task computation time vs. worst-case computation time,
   - ME: the maximum error in the estimation of task computation times.
2. System Parameters:
   - N_i: Node i,
   - R_i: Resource i,
   - MD: Message Delay from One Node to Another,
   - R: System-wide Non-periodic Task Arrival Rate,
   - L: System Load,
   - B: System-wide Node Balance Rate.
3. Scheduling Algorithms:
   - Non-Cooperative algorithm,
   - Random Scheduling Algorithm,
   - Focussed Addressing,
   - Bidding,
   - Flexible: Combines Bidding and Focussed Addressing Algorithms,
   - PSI: Perfect State Information Algorithm.
4. Algorithm Parameters
   - WL: Window Length,
   - FAS: Focussed Addressing Surplus,
   - SGS: System-wide Guarantee Surplus,
   - MB: Minimum Bid — A bid less than this value will not be sent,
   - HB: High Bid — For a task, if a node receives a bid higher than this value, the task will be awarded to the bidder immediately.
5. Performance Metrics
   - G: Task Guarantee Ratio,
   - G_{NP}: Non-Periodic Task Guarantee Ratio,
   - $G_{NP-local}$: Local Non-Periodic Task Guarantee Ratio,
   - $G_{NP-remote}$: Remote Non-Periodic Task Guarantee Ratio.

1 INTRODUCTION

Distributed real-time systems are becoming more prevalent in applications such as avionics, process control, and command and control systems. These applications contain many tasks that have execution deadlines that must be met. Tasks in these applications are typically categorized as being *critical*, *essential* and *nonessential*. Critical tasks are defined as those which must meet their deadlines under all circumstances, otherwise the result could be catastrophic. If essential tasks miss their deadlines, the performance of the system will seriously degrade. Nonessential tasks are defined as those whose deadlines if missed will not affect the system in the near future but may have an effect in the long term. Maintenance and bookkeeping activities fall in this category. Resources needed to meet the deadlines of critical tasks are typically preallocated. Also, these tasks are usually statically scheduled such that their deadlines will be met even under worst-case conditions. Strategies for dynamically scheduling essential tasks in a distributed system is the subject of this paper.

Many solutions used today for scheduling tasks are static in that they assume complete and prior knowledge of *all* tasks and make static scheduling and allocation decisions for the tasks. These solutions suffer from inflexibility and poor resource utilization. On the other extreme are solutions that assign priorities to tasks and design the system to perform preemptive priority-based scheduling. This approach suffers from a number of problems: Firstly, one figure, namely a task's priority, has to reflect a number of characteristics of the task including its deadline and level of importance. This assignment is error-prone and causes several well known anomalies because deadline and importance are not always compatible. Secondly, that a task has missed its deadline is known only when the deadline occurs. This does not allow time for any corrective actions. Thirdly, priority scheduling (as commonly defined) only addresses the cpu resource. This is a mistake. What value is there to immediately scheduling a task with a close deadline if the first thing that the task does is ask for a locked resource and therefore must wait? What is required is an integrated approach to cpu scheduling and resource allocation. Our work addresses this very important need.

The scheduling decisions made by the schemes discussed in this paper are based on task deadline *and* resource requirements. Also, the notion of *guarantee* underlies all scheduling decisions: When a task arrives at a node, the local scheduler at that node attempts to guarantee that the task will complete execution before its deadline, on that node. If the

attempt fails, the scheduling components on individual nodes cooperate to determine which other node in the system has sufficient resource surplus to guarantee the task. If such a node is not found, corrective action can be attempted before the deadline is missed. This guarantee based scheme significantly improves the predictability and fault tolerance properties of the system.

Designing good distributed scheduling algorithms that meet task deadlines is our goal. The performance metric used to evaluate the competing algorithms is the *guarantee ratio*: The ratio of the number of tasks which can be guaranteed to complete before their deadlines compared to the number of tasks which are invoked [1]. The goal of the scheduling algorithms presented in this paper will be to *maximize the guarantee ratio.*

Due to the real time constraints on tasks, the scheduling algorithm itself should be very efficient. That is, *the scheduling delay must be minimized* and the *scheduling overheads incurred by the system should be minimized.* This implies that the decisions, such as whether a task can be completed before its deadline as well as which node the task should be sent, must be made efficiently. The problem of determining an optimal schedule even in a multiprocessor system is known to be NP-hard [GRAH79]. A distributed system introduces further problems due to communication delays. All of these factors necessitate a *heuristic* approach to scheduling.

Our basic strategy for scheduling hard real time tasks in a loosely coupled distributed system is as follows: Assume that when a task arrives at a node of a distributed system, the task's deadline, computation time, and resource requirements are known. When the task arrives, the scheduler component local to that node decides if the new task can be *guaranteed* at this node. The guarantee means that no matter what happens (except failures) this task will finish execution by its deadline, and that all previously guaranteed tasks will still meet their deadlines [2]. If the new task cannot be guaranteed locally, then the scheduling components on individual nodes cooperate to determine whether another node has sufficient surplus in all those resources required by the task to guarantee the task and if such a node exists, the task is sent to that node; otherwise the task is rejected.

Algorithms for local guarantee under resource and timing requirements have already

[1] If tasks belong to different categories, such as essential and nonessential, or have different levels of importance, it is possible to define a suitable *weighted guarantee ratio.* We do not consider this case in this paper.

[2] If a task's execution has to be guaranteed in spite of failures, a guarantee should be obtained from multiple nodes. We do not discuss such multiple guarantees in this paper.

been studied [ZHAO87a, b] and will not be repeated here. Rather, this paper focuses on the algorithms for selecting a *remote node* to which a task should be sent when the task cannot be guaranteed locally.

A number of approaches are possible for distributed scheduling. A simple approach, called the *random scheduling* algorithm sends the task to another randomly selected node. Two other approaches, utilizing system state information, may be taken for the cooperation among nodes to select a remote node [RAMA84 and STAN85]: *bidding* and *focussed addressing*. In focussed addressing, a node with the highest estimated surplus is selected. In bidding, a node is selected if the node offers the best bid. Both bidding and focussed addressing utilize information on the state of other nodes and hence incur more overheads than random scheduling. The communication costs involved in bidding are high, but selection is made based on relatively accurate state information of nodes. Focussed addressing entails less communication costs and delay than bidding, though the use of incomplete, inaccurate, and out-of-date state information, increases the risk of making wrong decisions. In addition to these algorithms, a *flexible algorithm* that combines bidding and focussed addressing is described and studied. It is designed to reap the benefits of both and to overcome the shortcomings inherent in using each by itself.

We also compare these algorithms with two baseline algorithms. One is the *non-cooperative* algorithm. Here a task that is not guaranteed locally is not sent to any other node. The second is the *perfect-state-information* algorithm. This algorithm behaves exactly like the bidding algorithm, but it incurs no communication overheads.

One major contribution of this work is that it extends the state of the art in scheduling for real-time systems because it deals with algorithms that explicitly take into account the deadlines and resource needs of tasks. Besides proposing these algorithms, a major contribution of this paper is that the algorithms are evaluated to determine their effectiveness for scheduling tasks with hard-real-time constraints and general resource requirements. The simulations compare the algorithms different loads, different communication delays, different laxities and different loads at the nodes in the system which causes an unbalanced system if no distributed scheduling is performed. The effects of a computation time reclaim policy that several of the algorithms have, as well as the effects of erroneously specifying worst case computation times of tasks (which is one of the inputs to the scheduling algorithms) are also studied.

Most related work on scheduling tasks with hard-real-time constraints is restricted to CPU resources only [MUNT70, LIU73, DERT74, JOHN74, GARE75, MOK78, and TEIX78]. Perhaps the only exception is the work reported in [LEIN86] wherein, given the general resource requirements of each task, the worst-case response time of each task is determined. In Leinbaugh's model, a task is divided into multiple segments and the segments of a task can be executed concurrently on different nodes. His approach is useful at system design time to statically determine the upper bounds on response times.

In other work, Efe [EFE82] and Ma [MA84] use *heuristic approaches* for related task allocation problems. According to Lenat [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. In our algorithm, both types of heuristics are used to prevent the exponential search problem [ZHAO87a].

The remainder of this paper is organized as follows: Section 2 defines the system model adopted in this paper. Section 3 describes the algorithms for dynamic scheduling in hard-real-time distributed systems while Section 4 discusses the details of the steps involved in bidding and focussed addressing. The simulation results are presented and discussed in Section 5. Section 6 summarizes the work.

2 THE SYSTEM MODEL

There are n nodes, N_1, N_2, ..., N_n in a loosely coupled distributed system. Let each *node* contain a set of ***distinct*** resources, R_1, R_2, ..., R_r. A resource is an abstraction and can include CPU, I/O devices, files, data structures, 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. Some resources can be (simultaneously) shared by multiple tasks while others, such as a CPU, have to be assigned exclusively to one task. Further, if a sharable resource, such as a file, is

modified by a task, the resource should be exclusively assigned to the task.

A *task* is a scheduling entity and its execution cannot be preempted. The following characteristics of a task, T, are assumed known when it arrives:

- the worst case computation time, C(T); Tasks in real-time system have to be designed so that the difference between their worst-case and normal execution times is not large. Otherwise, when resources are assigned to a task for its worst case execution time, poor resource utilization will result. In this regard, a dynamic scheduling scheme has advantages since based on the input parameters of a dynamically invoked task, a lower worst-case computation time can be determined (compared to a statically determined worst-case computation time).
- The deadline, $D(T)$, by which the task must complete; (The laxity of a task T is defined to be $(D(T) - C(T) - current\_time)$.
- The resource requirements of the task. It is assumed that a task needs all its resources throughout its execution. A task will request at least one active resource and zero or more passive resources.

There are two types of tasks: *nonperiodic tasks* and *periodic tasks*. A *nonperiodic task* arrives at any node dynamically and has to be executed before its deadline. An instance of a *periodic task* with period P should be executed once every P units of time. Periodic tasks are assigned to nodes at system initialization time and their timely execution is guaranteed when nonperiodic tasks are scheduled. Since periodic tasks remain on the node where they are initially assigned, this paper focusses on the distributed scheduling of nonperiodic tasks. The effect of periodic tasks on local scheduling is however taken into account.

In addition to resource requirements and timing constraints, tasks in real-time systems are also characterized by their *priority* and *precedence constraints*. The priority of a task encodes its *level of importance* relative to other tasks. Precedence constraints enter the picture when tasks communicate or when a complex task is viewed in terms of a number of subtasks related by precedence constraints. Whereas distributed scheduling of tasks with precedence constraints is the subject of [CHEN86] and prioritized tasks are considered in [BIYA88], this paper focuses on tasks that are independent and have equal priority. This focus was chosen in order to carefully study a set of algorithms that vary in their complexity

but have general applicability. Consideration of precedence and priority would have added to the (already) large number of variables that can affect the results.

On each node, there are 3 components involved in task scheduling: The *local scheduler* handles scheduling of tasks that arrive at a given node. The *dispatcher* invokes the next task to be executed based on the schedule determined by the local scheduler. The *global scheduler* interacts with the schedulers on other nodes in order to perform distributed scheduling.

Nodes are connected by a communication network. The exact topology of the network is not important even though depending on the topology an algorithm could be optimized. Some of the algorithms implicitly take topology information into account, for instance, in determining the nodes to which local state information is sent.

3 ALGORITHMS FOR GLOBAL SCHEDULING

When a local task, T, arrives at a node N_i the *local scheduler* of N_i is invoked to try to guarantee the newly arrived task on the node. If the task can be guaranteed, it will be added to the *schedule* which contains all the guaranteed tasks on the node. Details of the local scheduling algorithm can be found in [ZHAO87a, b]. This section discusses five algorithms for dealing with a task that is not guaranteed locally.

The first plausible algorithm is the *Non-Cooperative algorithm* (NC). When a task cannot be guaranteed locally, it is rejected. No attempt is made to send the task to other nodes. One can see that if all nodes are heavily loaded, a non-cooperative strategy is the best. This algorithm is used as a baseline in the simulation studies.

The second algorithm is the *Random Scheduling Algorithm* (RSA). In this algorithm, when a task cannot be locally guaranteed, the node sends the task to a *randomly* selected node. The advantage of this algorithm is that it uses minimum communication overhead to determine the node to which a task should be sent. The disadvantage is that it is easy to send a task to an *improper* node because of the randomness.

A more informed choice can be made by a node if it has information about the state of

other nodes, in particular, about the resources available on a node as well as their surplus resources. The three algorithms to be discussed next consider a node for sending a particular task to only if that node has the resources needed by this task. Among the nodes that meet this criterion, a node is chosen based on the surplus information.

Each node periodically calculates the *node surplus* and sends it to a subset of the nodes in the system. 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 from the external environment and not from other nodes. A node's surplus is a vector, with one entry per resource on that node. Each entry indicates the total amount of time, in a (past) window, during which a resource is not used by the local tasks. The window is a time interval $[(t - WL), t]$ where t is the current time, and WL is the length of the window which is adjustable parameter of the algorithm. For example, within the recent window, suppose 400 is the sum of the length of all the time intervals during which a resource R on a node is not used by any task. Then the surplus on that node is said to be 400. WL should not be too short – in this case it may reflect the transient behavior of a node, and not too long – in this case it may not reflect the changes that are of legitimate interest to other nodes.

A node sorts other nodes according to the number of tasks received from them that were guaranteed on this node in the past time window. Then, according to this sorted node list, the node selects a subset of nodes to send information on 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. Hence, the nodes which recently sent more tasks to this node will more likely be selected. The above strategy minimizes the overheads of exchanging surplus information. One effect of this is that not all nodes will have the same state information about other nodes. If the network is small, the surplus information can be sent to all the other nodes.

Recall that a node knows the surplus of a given resource on other nodes. Thus, knowing the resources needed by a task and the computation time of the task the node can determine whether one or more nodes are in a position to meet the task's needs. The three algorithms discussed next differ in the way they select the node to which the task should be sent.

The *Focussed Addressing* (FA) algorithm works as follows. When N_i has a task that is not locally guaranteed, it determines the node with the highest surplus in the resources

needed by the task. If this surplus is greater than *Focussed Addressing Surplus (FAS)*, a tunable system parameter, the task is immediately sent to that node. If no such node is found, the task is rejected.

In the *Bidding* algorithm, k nodes with sufficient surplus in the resources needed by this task are selected. The value of k is chosen to maximize the chances of finding a node for the task. A request-for-bid message is sent to these nodes. When a node receives the request-for-bid message, it calculates a bid, indicating the likelihood that the task can be guaranteed on the node, and if it is higher than MB, the minimum required bid, sends the bid to the node which issued the request-for-bid. After receiving the bids, N_i sends the task to the node which offers the best bid. If there is no good bid available for the task, it is assumed that no node in the network is able to guarantee the task.

Various details of the bidding and focussed addressing algorithms are given in the next section. We now provide an overview of the *flexible algorithm*. As will be clear later, the *bidding algorithm* and the *focussed addressing algorithm* are special cases of this flexible algorithm.

- N_i selects k nodes with sufficient surplus in the resources needed by this task. If the largest value of the surplus of these k nodes is greater than FAS, then the node with that surplus is chosen as the *focussed* node. If a focussed node is found (see section 4.2), the task is immediately sent to that node. In addition to sending the task to the focussed node, node N_i sends in parallel, a request-for-bid message to the remaining $k - 1$ nodes. The request-for-bid message also contains the identity of the focussed node if there is one.

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

- When a task reaches a focussed 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 are evaluated and the task is sent to the node responding with the "highest bid". A message about whether and where the task is finally guaranteed is sent to the original node. The original node then modifies its surplus information about other nodes accordingly.

- In case there is no focussed node, the original node will receive the bids for the task and will send the task to the node which offers the best bid.
- If the focussed node cannot guarantee the task and if there is no good bid available for the task, it is assumed that no node in the network is able to guarantee the task.

Note that if k is 1, the above algorithm behaves like the focussed addressing algorithm. If the parameter *Focussed Addressing Surplus* is set to be large, it behaves like the bidding algorithm. Thus, the flexible algorithm combines features from focussed addressing and bidding, utilizing them in an opportunistic manner. The advantage of the flexible algorithm lies in the fact that its choice of focussed addressing, bidding or both is made *on a per-task basis*. For example, for a particular task, if a focussed 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 converges to the bidding scheme. On the other hand, if the surplus of the focussed 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 converges to focussed addressing.

In all these algorithms, the action to be taken when a task is not guaranteed depends on the application requirements as well as on the characteristics of the task. If the task has sufficient laxity then another attempt at global scheduling may be made. However, this will increase the scheduling and communication overheads. In general, the invoker of a task that is not guaranteed may invoke the same task with an increased deadline or may invoke another task that produces less precise results but with lower computational costs.

4 DETAILS OF BIDDING AND FOCUSSED ADDRESSING

This section deals with the the estimation and proper use of node surplus, the heuristics for choosing focussed nodes, the strategies for making bids, and the evaluation of the bids. The details provided are in the context of the flexible algorithm. The simplifications needed

when focussed addressing or bidding algorithms are used by themselves should be obvious given the descriptions in the previous section.

4.1 Focussed Addressing and Requesting Bids

The global scheduler at each node of the distributed system is responsible for doing focussed addressing and requesting bids. For $j = 1, ..., n$ and $j \neq i$, the global scheduler on node N_i estimates

$$ES(T, j) = \text{number of instances of task T that node } N_j \text{ can guarantee.} \quad (1)$$

This estimation is made according to the node surplus information available on node N_i and provides a good indication of the likelihood of a site being able to guarantee a given task. The global scheduler then uses this estimate to decide whether or not to try focussed addressing and/or bidding.

For example, assume that the computation time of task T is 250. Suppose, Node N_s is estimated to have a minimum surplus of 400 on *each* of the resources needed by T in the time interval in which task T must run. We say that the surplus of N_i with respect to the resources needed by task T is 400. Then ES(T, s) = 400/250 = 1.6. The global scheduler on Node N_i sorts other nodes according to their ES(T, j), in descending order. The first k nodes are selected to participate in focussed addressing and bidding. The value of k is decided such that the sum of ES(T, j) of the k nodes is larger than or equal to SGS, the *System-wide Guarantee Surplus.* This is a tunable parameter of the system. If the first node N_f among the k nodes has its ES(T, f) larger than FAS, the *Focussed Addressing Surplus*, node N_f is selected as the focussed node. The task is immediately sent to that node. The remaining $k - 1$ nodes are sent request-for-bid messages in parallel, to handle the case where the focussed node cannot guarantee the task. A request-for-bid message includes information about the deadline and computation time of the task as well as the latest bid arrival time, i.e., time by which bids should reach the focussed/requesting node to be eligible for further consideration. The latest bid arrival time for a task T, L(T), is estimated as follows:

$$L(T) = D(T) - C(T) - (TD + SD) \quad (2)$$

where D(T) is the deadline of T, C(T) is the computation time of T, TD is the (network-wide) average transmission delay between two nodes, and SD is the average scheduling delay on a bidder node. Thus, on the average, at or before L(T) there will be sufficient time to send the task to a bidder node, for it to be scheduled there, and then be executed before its deadline.

System performance is sensitive to the values assigned to SGS and FAS. FAS should be such that the chance of a focussed node guaranteeing a task will be high. SGS should not be too high, otherwise too many messages will be transmitted in the network. It should not be too low since this may result in too many *remote* tasks (i.e., tasks that were not locally scheduled) not being guaranteed, because request-for-bid messages may not be sent to the nodes that can guarantee a task.

4.2 Bidding, Bid Evaluation, and Response to Task Awarded

When a node receives a request-for-bid message for a task, it calculates a bid for the task. The *bid* indicates the number of instances of the task the bidder node can guarantee. The calculation is done in two steps: First, an upper bound of the bid, Max-Bid is determinated by:

$$\text{Max-Bid} = \frac{\text{Min(Free Time of Each Resource Required by the Task)}}{\text{Computation Time of the Task}}. \qquad (3)$$

The free time is calculated to be the sum of the lengths of the free time slots between the estimated earliest task arrival time on this node and the task's deadline. The earliest arrival time is estimated in an optimistic manner to be the sum of current time, the minimum message delay in transmitting the bid and the minimum message delay in sending the task to the bidder. Max-Bid is also calculated optimistically to be the best possible bid that this node can make assuming ideal availability of resources that the task needs, i.e., assuming that all the time slots when a resource is idle appear together and that all the needed resources are concurrently idle.

The second step calculates the actual bid. In this step, a binary search between 0 and Max-Bid is performed. In each stage of the binary search, a given number of instances of task T are temporally inserted into the current schedule of this node, and it is checked

whether the inserted instances can also be guaranteed. The maximum number of instances of the remote task T that this node can actually guarantee without jeopardizing previously guaranteed tasks is obtained at the end of the search. This number, if above *minimum bid* (MB), a tunable parameter, becomes the bid. The bid is sent to the node selected for focussed addressing if there is one. Otherwise, the bid is sent to the original node which issued the request-for-bid message. 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 is not affected by the bid it makes. This implies that a node does not reserve the resources needed by the tasks for which it bids. Since a node will typically bid for multiple tasks and multiple bids will be received for a task, reservation of resources will result in pessimistic bids and hence may reduce system performance.

When a node receives a bid for a given task, and the bid is higher than *High Bid* (HB), a tunable parameter, the node awards the task to the bidding node immediately and all other bids for this task, that arrived earlier or may arrive later, are discarded. If all the bids that have arrived for a given task are lower than HB, the node postpones making the awarding decision until L(T), the latest bid arrival time of the task. At time L(T), the task will be awarded to the highest bidder if any.

When the awarded task arrives at the highest bidder, the local scheduler on that node is invoked to see if the task can be guaranteed. Note that the state of the node may have changed after making a bid and since resources needed by the task were not reserved, the task may or may not be guaranteed. If the task is not guaranteed, it is rejected.

Note that this algorithm requires five tunable parameters: WL, FAS, SGS, MB, and HB. Section 5.2 discusses these parameters in more depth.

5 EVALUATION OF THE ALGORITHMS THROUGH SIMULATION

In this section, simulation results are presented for the algorithms discussed in the previous sections. We will see how different variables, such as, the network communication delay, the

task laxity, the load distribution among nodes, and the task computation times affect the performance of the algorithms.

In practice, a designer may wish to design a system to meet some performance requirements. The simulation results may be used to determine the system configuration, choose the appropriate scheduling algorithm and determine the parameters of the algorithm in order to meet given requirements. Hence, section 6 concludes this paper with a characterization of the performance of the algorithms, specifically with respect to system load, load distribution among nodes, and communication delays.

5.1 Simulation Model

The simulation is made with a system having six nodes where each node has two active resources and three passive resources. (Were it not for the excessive CPU time and memory requirements needed for this simulation we would have experimented with a larger set of nodes and with more resources.) There is one periodic task per node. It has a period of 2000 time units and a computation time of 400 units. The periodic task needs all the resources.

Unless specified otherwise, both the computation time and laxity of nonperiodic tasks are normally distributed. The mean of tasks' computation time and laxity are 400 and 600 time units respectively with respective standard deviations being 100 and 300.

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 node loads. Let us define the term *system-wide nonperiodic task arrival rate* (abbreviated as R) as the sum of the nonperiodic task arrival rates of all the nodes. To normalize the arrival rate, in the rest of this section let us define the task arrival rate to be the average number of arrivals per 100 time units (100 is the mean task computation time). Because of this normalization, the specific value chosen for the computation time, namely 400, is not very important. Also, even though we have chosen the nominal value for the mean laxity to be 600, we also study the effect of different laxity values.

Each nonperiodic task requires at least one active resource and zero or more passive resources. A task's resource requirements are chosen randomly. A nonperiodic task requests each of the two active resources with a probability of 2/3 and each passive resource with a

probability of 0.5. (This choice is arbitrary. It effects the performance via its effect on the system load, L.) For a given system nonperiodic task arrival rate R, L is measured as the average load on the active resources. It can be determined by

$$L = (2/3)(R/n) + 0.20 \tag{4}$$

where n is the number of nodes. The first term gives the average load on an active resource of a node caused by the nonperiodic tasks. The second term is due to periodic tasks (since periodic tasks take 400/2000, i.e., 20%, of computation power in each node).

The simulation model assumes that the global scheduler and the local scheduler are executed on a specially designed co-processor dedicated to scheduling and other system support tasks. Such a co-processor isolates application tasks from external interrupts and from the overheads caused by the execution of kernel modules and thus contributes to predictable behavior. Since real-time systems are characterized by the specialized hardware, for example, to process information from sensors, dependence on an additional piece of specialized hardware is not unreasonable, especially because of the additional benefits that it offers. The presence of the co-processor does not imply the absence of the time delay involved in guaranteeing a task. These delays, proportional to the square of the size of the task set processed by the guarantee routine [ZHAO87a], are taken into account in the simulation studies.

Nodes communicate with each other through a *star* communication network (see Figure 1). Each channel is associated with a node. One end of a channel is connected to the node it is associated with. The other end of all channels are connected together.

Messages, sent by a node to another, will pass the sender's channel first, then the receiver's channel. Messages pass a channel in a pipelined fashion with only one message occupying a channel at a given time. When a message is in a channel, if another message needs to be transmitted through the channel (in either direction), the latter must wait until the first has left the channel. This situation is called a *conflict*. The total time for transmitting a message from one node to another without any conflict is denoted as *conflict-free message delay* (MD). Because a message from one node to another passes two channels, the time taken by a message to pass a channel is half of its message delay.

Communication delays are also incurred for transferring tasks. In real-time systems, the set of tasks that execute in the system are known before the system begins operation. Because of this, it is feasible to store the code of the task at one or more nodes (in particular,

those that have the resources needed by the task), or all nodes if the number of nodes is not large. Thus, when one node wants another to execute a task, it needs to provide the other with information identifying the task as well as inputs relevant to the task. In this case, sending the task implies sending this required information. Only when task code is not available at the receiving node is it necessary for the sender to send the code for the task as well.

For the purpose of the simulation studies, we assume that the delay involved in transferring a task to another node is equal to the message delay plus 10% of the computation time of the task. Whereas the choice of the figure 10% is arbitrary, a fixed percentage of the task's computation time can be justified: Since the sender of a task sends the data needed for the execution of a task, it is plausible to assume that the amount of data processed by a task is proportional to the amount of processing done by the task. Another, but perhaps less plausible explanation is that the longer a task executes, the larger its code is.

Overall, the above network topology, the communication protocol, and the models for the communication overheads were chosen for simulation because they accentuate the effect of communication overhead on the performance of the algorithms.

It is often the case that the performance of a distributed scheduling algorithm depends on the the differences in the node loads. In order to quantify the differences in the node loads, and for the sake of simplicity, let us assume that the nonperiodic task arrival rates of the six nodes forms an equal-rate sequence, and the rate is denoted as the *load distribution factor* (B). That is, if the load distribution factor takes a value of B, and if at the first node, on average, N tasks arrive per unit time, then the i-th node has an arrival rate of NB^{i-1}. 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, 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 node loads will be.

Clearly, the value assigned to the different tunable parameters will affect the system performance. This is attested by the simulation studies; results of only some of these are reported here due to space limitations. These studies indicate that the algorithms studied here are *relatively stable* in the sense that the percentage change in system performance for a given change in parameter value is very small. In practice, this implies that the system performance will not change drastically when the parameters do not deviate much from the "best value" – the value at which the best system performance is achieved for a given

application. We return to the issue of stability further at the end of Section 5.2.

For the simulation studies, the values for the tunable parameters are set as follows. Simulation studies (not reported here) indicated that an "optimum" WL value lies in the range of 30 - 70 times the mean task inter-arrival time, depending on task characteristics. So the window length is set to be 50 times the mean of the task inter-arrival times. Surplus information is transmitted to other nodes periodically where the period is also set to be 50 times the mean task inter-arrival times. Since the network has only six nodes, surplus information is sent to all the nodes in the network.

System-wide Guarantee Surplus (SGS) is 2.0. Focussed Addressing Surplus (FAS) is 1.2. Minimum Bid (MB) is 1.0, and High Bid (HB) is 2.0.

Each data point in the simulation results reported in the following sections indicates the average of three runs where each run was made to last for (at least) 3000 times the mean task inter-arrival time.

Before presenting the simulation results, let us define the following terms to describe the system's performance. Since meeting task deadlines is the goal of any hard real-time system, the guarantee ratio is a good measure of the performance of scheduling algorithms.

$$G = \text{System Task Guarantee Ratio} = \frac{\text{Total Number of Tasks Guaranteed}}{\text{Total Number of Task Arrivals}} \tag{5}$$

$$G_{NP} = \text{System Non Periodic Task Guarantee Ratio} = \frac{\text{Total Number of Non Periodic Tasks Guaranteed}}{\text{Total Number of Non Periodic Task Arrivals}} \tag{6}$$

$$G_{NP-local} = \text{System Non Periodic Local Task Guarantee Ratio} = \frac{\text{Total Number of Local Non Periodic Tasks Guaranteed}}{\text{Total Number of Non Periodic Task Arrivals}} \tag{7}$$

$$G_{NP-remote} = \text{System Non Periodic Remote Task Guarantee Ratio} = \frac{\text{Total Number of Remote Non Periodic Tasks Guaranteed}}{\text{Total Number of Non Periodic Task Arrivals}} \tag{8}$$

5.2 Comparison Of Focussed Addressing, Bidding and the Flexible Algorithms

The first set of simulation results show how the communication delay affects the performance (G) of the three algorithms that utilize state information.

In this simulation, three of the six nodes have equal loads which are higher than the remaining three nodes which also have equal loads. The load on the latter was set to be 0.3 times that of the former. Note that for a given system arrival rate, R, and the ratio of the node loads in the two sets, the arrival rate for each node can be determined. Three cases were tested:

1. The system load is light (the system nonperiodic task arrival rate, R, is equal to 3 per 400 time units, and the system load L = 0.53);
2. The system load is heavy (R = 6.0 and L = 0.87); and
3. The system is overloaded (R = 9.0 and L = 1.20).

We study the performance under different conflict-free message delays (MD) from 0, with increments of 20 time units, up to 200 time units. For each case, the performance of the flexible algorithm is compared with the bidding algorithm and the focussed addressing algorithm.

From Figures 2, 3, and 4, the following are observed:

- The communication overhead has explicit effect on the performance of the flexible algorithm as well as the bidding algorithm. For example, when the load is heavy, the performance of the flexible algorithm degrades by about 10% when MD increases from 0 to 200.
- The performance of the focussed addressing algorithm is not as sensitive to MD as the other two algorithms. But, when MD is small, the difference in the performances between the focussed addressing algorithm and the flexible algorithms is higher than when MD is large.
- The performance curve of the flexible algorithm follows the "envelope" of the curves for the bidding and the focussed addressing algorithms, indicating that the flexible

algorithm performs well over almost the entire range of MDs. In fact, for most communication delays, the flexible algorithm does better than bidding and focussed addressing. This is because, unlike these algorithms (that use one scheme for all tasks), the flexible algorithm chooses to do bidding, focussed addressing, or both on a per-task basis, depending on the system state and task characteristics at the time the choice is made.

- The performance of the bidding algorithm is close to the flexible one when MD is small (below 40). As MD increases, the difference in the performance between the two schemes increases.

- When MD is very high, focussed addressing is a better choice than bidding. This is because bidding involves more message traffic. Also, the overhead of processing the request-for-bid messages and bids affects the performance of the bidding algorithm. The performance of the flexible algorithm is equal to the focussed addressing algorithm for MD above 160, showing the adaptability of the flexible algorithm.

The flexible algorithm uses a number of tunable parameters: WL, FAS, SGS, HB, MB. Of these, FAS and MB are expected to have more impact on the performance than the others. This is because FAS is used to determine a focussed node, and MB determines whether or not a node is going to offer a bid. Figures 5 and 6 show the effect of these two parameters on the performance of the bidding, focussed addressing and the flexible algorithms. The figures show that with respect to these two parameters, the algorithms are *relatively stable*, that is, their performance does not vary widely when the parameter value is varied. In addition, while producing better performance, the flexible algorithm also displays better stability compared to bidding and focussed addressing algorithms. Similar observations have also been made for other parameters such as WL, SGS, HB. They are not reported here due to space limitations. We believe that 5 tunable parameters for a distributed scheduling algorithm that takes deadlines and resource allocation into account is not unreasonable. For example, the VAX VMS uni-processor scheduling algorithm is a multi-level feedback queue that has many tunable parameters including the base level priority of a task, the highest priority level, the rate at which a priority is increased for an I/O, the rate at which the priority is dropped for using up a whole time slice, the length of the time slice, etc.

Since the flexible algorithm performs better than bidding and focussed addressing based

algorithms, in the remainder of this section we focus on the flexible algorithm.

5.3 Comparison of Flexible Algorithm with the Baselines

We now compare the performance of the flexible algorithm with that of the non-cooperative algorithm and the *Perfect State Information scheduling algorithm* (PSI). The PSI algorithm behaves like the bidding algorithm except for the following differences: Communication overheads are zero, i.e., the system overheads to obtain surplus information is assumed to be zero and the cost of sending request for bids as well as bids is also assumed to be zero. The performance of this algorithm should be better than all practical algorithms, thus serving as an upper bound for a distributing scheduling algorithm.

We test three cases with B = 0.50, 0.75, and 1.00 respectively. In each case, the system nonperiodic task arrival rate, R, varies from 1.00 to 10.00. The MD is 30 time units. Figures 7, 8, and 9 plot the guarantee ratio G vs. system nonperiodic task arrival rate, R, for these 3 cases. Each figure depicts the performance of the flexible algorithm, the Non-Cooperative algorithm (NC) and the Perfect State Information algorithm (PSI).

These simulation results indicate that the performance of the flexible 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 load distribution factor.

5.4 Comparison of Flexible and Random Algorithms

We now show the simulation results that compare the performance of the flexible algorithm with that of the random scheduling algorithm. The simulation parameters are set as before.

Two cases were tested. In each case, the flexible scheduling algorithm and the random scheduling algorithm are studied with nonperiodic task arrival rate from 2 to 10 per 400 time units. In the first case, there are 4 nodes which are relatively heavily loaded, and 2 nodes which are relatively lightly loaded. In the second, there are 2 nodes relatively heavily loaded, 2 nodes moderately loaded, and another 2 nodes lightly loaded. MD is 30 time

units.

Figures 10 and 11 show the results:

- In general, the flexible algorithm performs better than the random focussed addressing algorithm, especially at moderate loads. This is expected, since the decisions of the flexible algorithm are made by using the network-wide surplus information, but in the random focussed addressing algorithm, they are made randomly.

- However, because the flexible algorithm uses much more communication than the random scheduling algorithm, the non negligible communication delay may hurt the flexible algorithm, resulting in a poor performance, especially when the system load is very high. However, the flexible algorithm contains mechanisms to avoid sending messages which are not necessary, for example, by choosing to do just focussed addressing. From the simulation results shown above, it can be seen that even in the cases when the system load is very high, the flexible algorithm performs better than the random scheduling algorithm. This fact indicates that the mechanisms in the flexible algorithm to prevent the transmission of unnecessary messages have a positive effect.

- When the system load is very low, the performance difference between the flexible algorithm and the random scheduling algorithm is negligible. This is because when the load is light, most of the nodes have enough surplus so that any node selected randomly is as good as any other. For a similar reason, when the nodes are well balanced and the system load is high, the random algorithm should perform as well as the flexible algorithm. When the system is overloaded (say $R > 8$), the performance difference between the flexible algorithm and the random algorithm is very small. The minimal communication overheads incurred by the random algorithm play an important role here.

It is important to note that the above results were obtained after including the overheads incurred by the flexible algorithm and thus its complexity has been taken into account. Even with this, it performs better than the simpler random scheduling though the performance difference is less than five percent in the cases studied. As was discussed in the introduction, in a hard real-time system, every task that misses its deadline can seriously degrade the

performance of the system. Hence even a small performance improvement may be considered significant in the context of hard real-time systems.

5.5 Effect of Task Laxity on the Performance of the Flexible Algorithm

This study examines how the laxity of tasks affects the performance of the flexible algorithm. Three nodes have heavier loads compared to the remaining three nodes. The load of the latter is 0.3 times that of the former. We study the performances of three cases:

1. The laxity is relatively small with a normal distribution having a mean of 300 and a standard deviation of 150, denoted as $N(300, 150^2)$;
2. The laxity is moderate with a distribution of $N(600, 300^2)$; and
3. The laxity is large with a distribution of $N(900, 450^2)$.

All the laxities are larger than 0. In each case, the system nonperiodic task arrival rate, R, varies from 1 to 9 per 400 time units. From the simulation results, (Figure 12), the following observations can be made:

- The task laxity affects the system performance. When the laxity increases from $N(300, 150^2)$ to $N(900, 450^2)$, the number of tasks guaranteed increases significantly. For example, when R = 5.0, the system task guarantee ratio increases substantially from 76.3% to 92.6%.
- The system performance, measured as the system task guarantee ratio, is not a linear function of the mean of the laxity. For example, when the nonperiodic task arrival rate is 5.0 and the mean of the laxity increases from 300 to 600, the system task guarantee ratio increases from 76.3% to 87.9%, i.e., 12.6%. While the laxity increases from 600 to 900, and the system task guarantee ratio increases from 87.9% to 92.6%, i.e., 4.7%. When the arrival rate is small, this increase is more obvious than when the rate is high.

MD = 10 time units; Non Periodic Task Arrival Rate = 3 per 400 time units				
B	$G_{NP-local}$	$G_{NP-remote}$	G_{NP}	G
0.100	0.428	0.488	0.916	0.940
0.300	0.588	0.342	0.930	0.951
0.500	0.748	0.192	0.940	0.957
0.700	0.870	0.091	0.961	0.973
0.900	0.926	0.042	0.969	0.978
1.000	0.927	0.029	0.956	0.969

Table 1: Effect of Node Balance — Case A

MD = 10 time units; Non Periodic Task Arrival Rate: 6 per 400 time units				
B	$G_{NP-local}$	$G_{NP-remote}$	G_{NP}	G
0.100	0.280	0.524	0.804	0.837
0.300	0.408	0.409	0.816	0.847
0.500	0.519	0.304	0.823	0.853
0.700	0.639	0.200	0.839	0.867
0.900	0.708	0.156	0.864	0.888
1.000	0.722	0.138	0.860	0.884

Table 2: Effect of Node Balance — Case B

5.6 Effect of Load Distribution on the Flexible Algorithm

The purpose of this study is to examine how the differences in the loads of nodes affects the performance of the flexible algorithm.

Nine cases were studied (Cases A to I). For MD = 10 time units, the system load is varied from 3, 6, to 9 (Cases A to C). This is repeated for MD = 60 time units (Case D to F) and 120 time units (Cases G to I). In each case, the load distribution factor (B) takes values of 0.1, 0.3, 0.5, 0.7, 0.9, and 1.0.

The simulation results of these 9 cases are listed in Tables 1 to 9. For convenience of discussion, we further summarize Tables 1 to 9 in Table 10. Table 10 lists the differences between the best and worst performance in each case (under the column of "diff"), and points out when (in terms of load distribution factor, B) the maximum performance occurs

MD = 10 time units; Non Periodic Task Arrival Rate: 9 per 400 time units				
B	$G_{NP-local}$	$G_{NP-remote}$	G_{NP}	G
0.100	0.212	0.359	0.571	0.622
0.300	0.304	0.323	0.627	0.671
0.500	0.389	0.252	0.641	0.684
0.700	0.467	0.182	0.649	0.691
0.900	0.536	0.104	0.640	0.683
1.000	0.570	0.049	0.619	0.665

Table 3: Effect of Node Balance — Case C

MD = 60 time units; Non Periodic Task Arrival Rate: 3 per 400 time units				
B	$G_{NP-local}$	$G_{NP-remote}$	G_{NP}	G
0.100	0.427	0.436	0.863	0.902
0.300	0.587	0.308	0.895	0.926
0.500	0.750	0.162	0.912	0.938
0.700	0.873	0.056	0.929	0.950
0.900	0.929	0.022	0.950	0.965
1.000	0.929	0.010	0.939	0.957

Table 4: Effect of Node Balance — Case D

MD = 60 time units; Non Periodic Task Arrival Rate: 6 per 400 time units				
B	$G_{NP-local}$	$G_{NP-remote}$	G_{NP}	G
0.100	0.285	0.413	0.699	0.750
0.300	0.409	0.339	0.748	0.791
0.500	0.525	0.240	0.765	0.805
0.700	0.653	0.140	0.793	0.829
0.900	0.742	0.068	0.810	0.843
1.000	0.761	0.051	0.812	0.845

Table 5: Effect of Node Balance — Case E

MD = 60 time units; Non Periodic Task Arrival Rate: 9 per 400 time units				
B	$G_{NP-local}$	$G_{NP-remote}$	G_{NP}	G
0.100	0.218	0.192	0.410	0.480
0.300	0.311	0.214	0.525	0.582
0.500	0.417	0.163	0.580	0.630
0.700	0.503	0.108	0.611	0.658
0.900	0.566	0.058	0.624	0.669
1.000	0.584	0.030	0.614	0.661

Table 6: Effect of Node Balance — Case F

MD = 120 time units; Non Periodic Task Arrival Rate: 3 per 400 time units				
B	$G_{NP-local}$	$G_{NP-remote}$	G_{NP}	G
0.100	0.427	0.412	0.839	0.885
0.300	0.596	0.280	0.876	0.912
0.500	0.751	0.151	0.903	0.931
0.700	0.872	0.049	0.920	0.944
0.900	0.928	0.014	0.942	0.960

Table 7: Effect of Node Balance — Case G

MD = 120 time units; Non Periodic Task Arrival Rate: 6 per 400 time units				
B	$G_{NP-local}$	$G_{NP-remote}$	G_{NP}	G
0.100	0.284	0.363	0.646	0.706
0.300	0.403	0.314	0.718	0.766
0.500	0.533	0.212	0.745	0.788
0.700	0.655	0.117	0.772	0.811
0.900	0.758	0.032	0.790	0.826
1.000	0.773	0.022	0.796	0.832

Table 8: Effect of Node Balance — Case H

MD = 120 time units; Non Periodic Task Arrival Rate: 9 per 400 time units				
B	$G_{NP-local}$	$G_{NP-remote}$	G_{NP}	G
0.100	0.217	0.138	0.355	0.432
0.300	0.316	0.231	0.547	0.601
0.500	0.417	0.160	0.577	0.627
0.700	0.514	0.074	0.589	0.638
0.900	0.584	0.026	0.610	0.657
1.000	0.598	0.010	0.608	0.655

Table 9: Effect of Node Balance — Case I

	$G_{NP-local}$		$G_{NP-remote}$		G_{NP}		G	
Case	diff	max	diff	max	diff	max	diff	max
A	0.499	1.0	0.459	0.1	0.045	0.9	0.038	0.9
B	0.448	1.0	0.386	0.1	0.060	0.9	0.051	0.9
C	0.348	1.0	0.310	0.1	0.078	0.7	0.069	0.7
D	0.502	1.0	0.426	0.1	0.087	0.9	0.063	0.9
E	0.476	1.0	0.362	0.1	0.113	1.0	0.095	1.0
F	0.366	1.0	0.184	0.3	0.214	0.9	0.189	0.9
G	0.501	1.0	0.389	0.1	0.103	1.0	0.085	1.0
H	0.474	1.0	0.341	0.1	0.150	1.0	0.126	1.0
I	0.387	1.0	0.221	0.3	0.255	0.9	0.225	0.9

Table 10: Summary of Case A - Case I

(under the column of "max").

From these tables the following observations can be made:

- The node balance has a great effect on the local nonperiodic task guarantee ratios. The difference due to the node imbalance may be as high as 50% (Cases A, D, and G). That is, in the case where the system is heavily unbalanced, the tasks guaranteed locally may be 50% less than in the case where the system is well balanced.

- Consequently, the node balance affects $G_{NP-remote}$ the percentage of nonperiodic tasks which are remotely guaranteed. This percentage typically has its maximum at the load distribution factor of 0.1. This is because at that time some nodes are very heavily

loaded while others are very lightly loaded and hence the system has the most number of tasks which cannot be guaranteed locally. However, it is interesting to see that sometimes $G_{NP-remote}$ does not have its maximum at the load distribution factor of 0.1, but at the load distribution factor of 0.3. This happens when the system is heavily loaded and the MD is not small (Cases F and I). One explanation is that when the system is heavily unbalanced and MD is not small, too many remote tasks and the messages for scheduling them are transmitted in the network such that the overall network delay increases.

- The node balance has an explicit effect on system performance — the total task guarantee ratio, G. For example, in Case I it causes a difference of 22.5% in the total task guarantee ratios.

- However, it is interesting to notice that in only 3 out of 9 cases the maximum performance is achieved at the best balanced situation (node load distribution factor = 1.0). This is because when the nodes are slightly unbalanced, the benefit, gained from sending nonperiodic tasks which cannot be locally guaranteed to other nodes (where their resource requirements match with other tasks there), may be larger than the loss due to the imbalance. Also, since resource requirements are also considered, a balanced arrival rate across all nodes does not imply a balanced load on all resources of a node. The tables show that in most cases as soon as the load distribution factor, B, reaches 0.7, the system performance is very close to the maximum it can achieve. Consequently, a perfect node balance is not necessary with the flexible algorithm.

The results show that although a perfect node balance is not necessary, an extremely unbalanced situation should be avoided. To prevent such a situation, algorithms discussed in this paper can be augmented to balance loads by transfering guaranteed tasks from highly loaded nodes to nodes that are lightly loaded. Of course, such transfers should be attempted only after ensuring that the task will remain guaranteed even after the transfer.

5.7 Effect of Task Computation Times on the Flexible Algorithm

In the simulation discussed so far, it was assumed that no matter what the real computation time is, the system allocates the resources needed by the task for the worst case computation

time. Obviously, under this policy, resources are wasted if a task completes before its worst case computation time.

In the simulation results reported in this sub-section, each time when a task finishes (in many cases, before its worst case computation time) the resources are allocated to the task which needs these resources and is waiting to be dispatched. The simulation results show that the system performance can be improved significantly under this *reclaim policy*.

Tasks are normally guaranteed with respect to their worst case computation times. In practice, it is often very difficult to predict precisely the computation time for a task. That is, the estimation of task computation times in practice always has some error. Consequently, the worst computation time claimed by the task must be the estimated computation time plus the maximum error under the estimation. We will show by simulation that although the maximum error affects the system performance, the flexible algorithm can tolerate a certain amount of this error.

MD in the simulation is set to be 60 time units. B = 0.6.

Effect of Reclaim Policy

In this part of the simulation, we will see how the reclaim policy improves the system performance under different cases.

The worst case computation for a nonperiodic task is a random number from $N(400, 100^2)$. The real computation time of a task is assumed to be a uniform random number taken from the interval which is centered by the mean computation time and right-bounded by the claimed worst case computation time. For example, if the mean computation time = 360 and the claimed worst computation time = 400, then the real computation time is randomly taken from [320, 400].

For a given worst case computation time of a task, the mean computation time of the task is determined by a parameter K which is defined as

$$K = \frac{\text{Mean of Non Periodic Task Computation Time}}{\text{Worst Case Non Periodic Task Computation Time}} \tag{9}$$

Note that when K equals 1.0, the system reflects the original policy. To testify that the system performs better under the reclaim policy, we should show that the system guarantee ratio, G, is higher when $K < 1$ than when $K = 1$.

We test 3 cases for the system nonperiodic task arrival rates to be 3, 6, and 9 per 400

time units respectively. In each case, we let K vary from 0.5 to 1.0. The simulation results are shown in Figure 13. As expected, the new policy does improve the system performance, i.e., the system task guarantee ratio, G. When the system load increases, the improvement increases. In the overload case, if K is 0.5, the improvement can be as high as 14%.

It should be pointed out that the flexible algorithm can be enhanced to take into account the fact that many tasks do not execute up to their worst-case computation times. For instance, higher bids can be made by a node if it expects guaranteed tasks to finish sooner that is indicated by their worst-case computation times. The resulted reported here do not include the effect of any such enhancement.

Effect of the Maximum Error

Here, we will examine how the maximum error in estimating the computation time for a task, may effect the system performance. We assume that it is possible to know the value of the maximum error rate.

For simplicity, we fix the mean computation time for all tasks to be 400 time units. The worst case computation time, which a task claims when it arrives, will be (1 + maximum error rate) * 400. The actual computation time of a task is randomly chosen from the interval of [(1 - maximum error rate)*400, (1 + maximum error rate)*400]. The reclaim policy is used.

We test 3 cases with the system nonperiodic task arrival rate R being 3, 6, and 9 per 400 time units respectively. In each case, we let the maximum error rate (ME) vary from 0.0 to 1.0. From Figure 14, the following can be observed.

- The maximum error in estimating computation time of tasks can affect the system performance. When the error increases, the system performance decreases in all three cases. Specifically, when the system load is high (heavy load or overload) and the error is large ($\sim$ 50%) , the system performance will degrade significantly. However, when the maximum error is small (< 20%), in all 3 cases the task guarantee ratios reduce by at most 1.5%, compared with the error rate of 0. We conclude that the flexible algorithm can tolerate small amounts of error of this kind.

- The difference between the maximum and minimum guarantee ratios for light load is 93.9% − 90.1% = 2.8% ; for heavy load is 80.1% − 71.6% = 8.5%; and for overload is 64.1% − 57.1 = 7.0%. It is interesting to observe that when the system load is heavy,

the effect of the maximum error is larger than when the system is lightly loaded or overloaded. This is probably because when the system load is light, a task may be guaranteed anyway; consequently the effect is small. When the system load is very heavy (overload), if a task finishes before its claimed worst case computation time, it is more likely that there are tasks waiting to be executed, resulting in the noted performance improvement.

One aspect of algorithm behavior that merits attention is the fairness of a scheduling algorithm to tasks of different lengths. Data obtained in the course of the simulation studies indicate that distributed scheduling algorithms that utilize resource surplus information are likely to have better success with tasks that have smaller computation times than algorithms that do not use state information, for example, random scheduling. This is to be expected: Since resources are almost always scarce, in general, it is easier to find a node for a task that needs a set of resources for a shorter duration than another task. One way to "correct" this bias may be to choose a local scheduling algorithm that has a bias towards tasks that have long execution times. Clearly, the issue of bias requires to be studied further.

6 CONCLUSIONS

Many interesting real-time scheduling problems are known to be NP-hard [GRAH79]. The problems are further complicated when, in addition to computation times and deadlines of tasks, their resource requirements need to 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 discussed and evaluated different heuristic schemes for solving this problem. The algorithms perform on-line scheduling of tasks that arrive dynamically and handle nonperiodic tasks even in the presence of periodic tasks, i.e., tasks that are known to occur at regular intervals. They work in conjunction with a heuristic local scheduling algorithm – one that determines a schedule for tasks that will execute on a node given that those tasks have dealines to meet and require multiple resources (not just the cpu)[ZHAO87a]. The cooperation among the nodes, needed when a node is unable to guarantee a task, can

occur in a number of ways as illustrated by the algorithms discussed in this paper: On the one extreme is an algorithm that does not cooperate with other nodes. On the other extreme is an algorithm that utilizes bidding and focussed addressing techniques in a flexible manner.

Bidding and focussed addressing techniques discussed here are refined forms of the traditional source-initiated and server-initiated scheduling techniques [FARB72, SMIT82, and WANG85]. The two most important refinements are the manner in which the two techniques are combined in the flexible algorithm to give the benefits of both, and how timing and general resource requirements are included. Neither of these has been done before. Other differences also exist. For example, in focussed addressing, a source node regularly sends to other nodes its surplus information. A large surplus is an implicit request for tasks. This is unlike the approach in [WANG85] where a node requests for a task only when it is idle. The surplus information is also used in bidding. Using this information, a server node selects a subset of nodes, to send the request-for-bid messages, eliminating the transmission of unnecessary messages and hence reducing the network traffic. As shown by the simulation studies, properly combining these two improved techniques has the potential to make use of their positive features while overcoming their shortcomings, resulting in an algorithm with improved performance.

Results of the simulation studies can be used to make several observations regarding the relative performance of the various algorithms with respect to the system state, specifically, the system load, the load distribution among nodes, and the communication delays.

- Distributed scheduling inproves the performance of a hard real-time system. This is attested by the better performance of the flexible algorithm compared to the non-cooperative baseline under all load distributions.

- The performance of the flexible algorithm is better than both the focussed addressing and bidding algorithms. However, the performance difference between the bidding algorithm and the flexible algorithm under small communication delays is negligible. The same can be said about the performance difference between the focussed addressing algorithm and the flexible algorithm at large communication delays.

- The random algorithm performs quite well compared to the flexible algorithm, especially when system load is low as well as when system load is high and the load

is unevenly distributed. Under moderate loads, its performance falls short by a few percentage points which may be significant in a hard real-time system.

In summary, no algorithm outperforms all others in all system states. Though the flexible algorithm performs better than the rest in most cases, it is more complex than the other algorithms. Thus, if a given situation occurs most often in a given hard real-time system and in that situation one of the other algorithms performs as well as the flexible algorithm, that algorithm should be adopted for distributed scheduling in the system.

On the other hand, a hard real-time system whose system state changes with time should benefit from an adaptive distributed scheduling strategy. Results of the simulation studies can be used to develop a *collection of rules* that can be used in such an adaptive strategy. For example, when the communication delays are small, a bidding scheme works better than the rest. In [RAMA87] the possibility of a higher level of control, called *meta-level control* is discussed to improve the performance and enhance the *adaptability* of scheduling schemes. Such meta-level control involves monitoring and characterizing system state, and determining the choice of the algorithm appropriate for current system conditions. Further work is in progress to investigate the efficacy of meta-level control.

References

[BIYA88] S. Biyabani, "The Integration of Deadlines and Criticalness in Hard Real-Time Scheduling," Masters Thesis, Univ. of Mass., May 1988.

[CHEN86] S. Cheng, J.A. Stankovic, and K. Ramamritham, "Dynamic Scheduling of Groups of Tasks with Precedence Constraints in Distributed Hard Real-Time Systems", *Real-Time Systems Symposium*, Dec 1986.

[DERT74] Dertouzos, M., "Control Robotics: The Procedural Control of Physical Process", *Proc. of the IFIP Congress*, 1974.

[EFE82] Efe, Kemal, "Heuristic Models of Task Assignment Scheduling in Distributed Systems", *IEEE Computer*, June 1982.

[FARB72] Farber, D.C., and Larson, K.C., "The Distributed Computer System", *Proc. Symp. Comput. Commun. Networks and Teletraffic*, 1972.

[GARE75] Garey, M.R. and Johnson, D.S., "Complexity Results for Multiprocessor Scheduling under Resource Constraints", *SIAM Journal of Computing*, 4, 1975.

[GRAH79] Graham, R.L. et al, "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey", *Annals of Discrete Mathematics*, 5, 1979.

[JOHN74] Johnson, H. H. and Madison, M.S., "Deadline Scheduling for a Real-Time Multiprocessor", NTIS (N76-15843), Springfield, VA, May 1974.

[LEIN86] Leinbaugh, D.W. and Yamini, M., "Guaranteed Response Times in a Distributed Hard Real-Time Environment", *IEEE Trans. Software Engineering*, Dec. 1986, pp 1139-1144.

[LEN83] Lenat, Douglas B., "Theory Formation by Heuristic Search — The Nature of Heuristics II: Background and Examples", *Artificial Intelligence*, 21 (1983).

[LIU73] Liu, C. L., and Layland, J., "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, Vol. 20, No. 1, Jan. 1973.

[LIVN82] Livny, M., and Melman, M., "Load Balancing in Homogeneous Broadcast Distributed Systems", *Proc. ACM Computer Network Performance Symposium*, April 1982.

[MA82] Ma, Richard P., Lee, Edward Y.S., and Tsuchiya, Masahiro, 'A Task Allocation Model for Distributed Computing Systems", *IEEE Transactions on Computers*, Vol C-31, No. 1, Jan. 1982.

[MA84] Ma, Richard P., "A Model to Solve Timing-Critical Application Problems in Distributed Computer Systems", *IEEE Computer*, Jan. 1984.

[MOK78] Mok, A.K. and Dertouzos, M.L., "Multiprocessor Scheduling in a Hard Real-Time Environment", *Proc. of the Seventh Texas Conference on Computing Systems*, Nov 1978.

[MUNT70] Muntz, R. R., and Coffman, E. G., "Preemptive Scheduling of Real-Time Tasks on Multiprocessor Systems," *Journal of the ACM*, Vol. 17, No. 2, April 1970.

[RAMA84] Ramamritham, K., and Stankovic, J. A., "Dynamic Task Scheduling in Distributed Hard Real-Time Systems", *IEEE Software*, Vol. 1, No. 3, May 1984.

[RAMA87] Ramamritham, K., J. Stankovic, W. Zhao, "Meta-Level Control in Distributed Real-Time Systems," *Conference on Distributed Computing Systems*, Sept. 1987.

[SMIT80] Smith, R.G., "The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver", *IEEE Transactions on Computers*, Vol. C-29, No. 12, Dec. 1980.

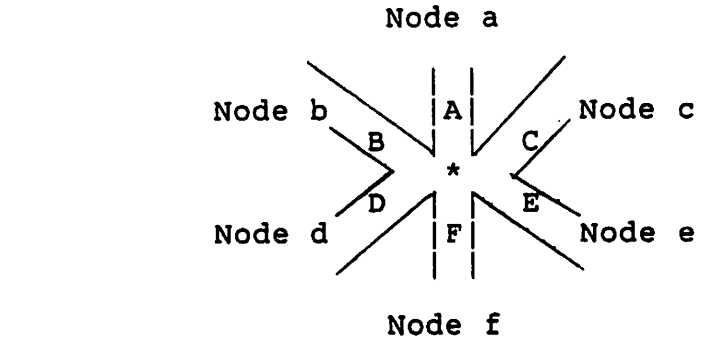
[STAN85] Stankovic, J.A., Ramamritham, K., and Cheng, Shengchang, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-time Systems", *IEEE Transactions on Computer*, December 1985.

[TEIX78] Teixeira, T., "Static Priority Interrupt Scheduling", *Proc. of the Seventh Texas Conference on Computing Systems*, Nov. 1978.

[WANG85] Wang, Y., and Morris, R., "Load Sharing in Distributed Systems", *IEEE Transactions on Computers*, Vol. C-34, No. 3., March 1985.

[ZHAO87a] Zhao, W., Ramamritham, K., and Stankovic, J.A., "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems", *IEEE Transactions on Software Engineering*, May 1987.

[ZHAO87b] W. Zhao and K. Ramamritham, "Simple and Integrated Heuristic Algorithms for Scheduling Tasks with Time and Resource Constraints", *Journal of Systems and Software*, 1987.



Nodes: a, b, ..., d Channels: A, B, ..., F

Figure 1: Communication Network

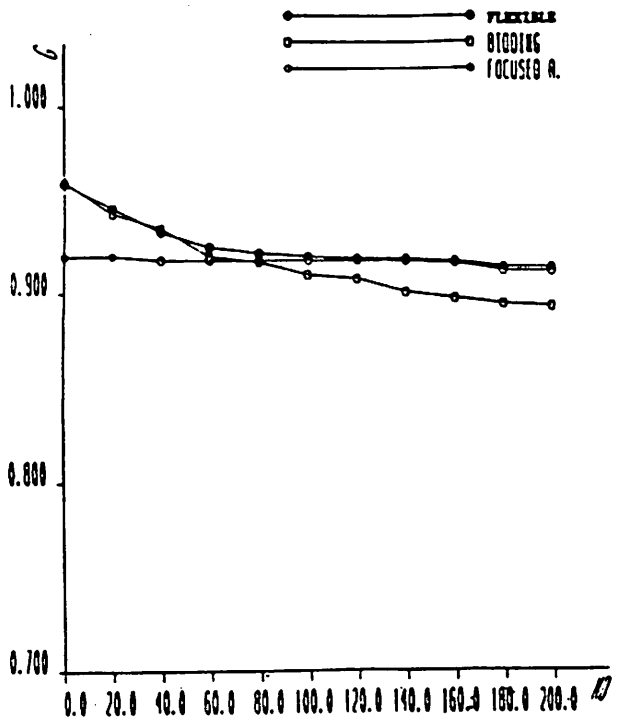


Figure 2: Performance of bidding, focussed addressing and flexible algorithms — Light Load

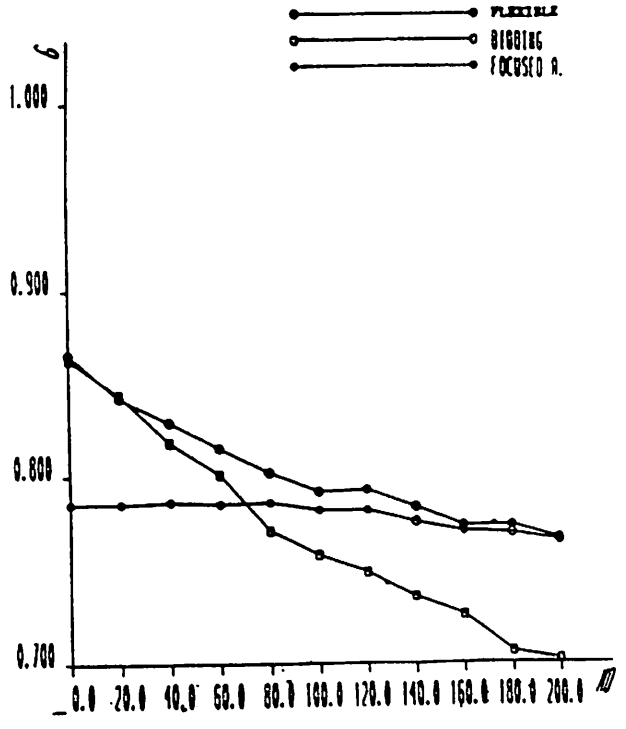


Figure 3: Performance of bidding, focussed addressing and flexible algorithms — Heavy Load

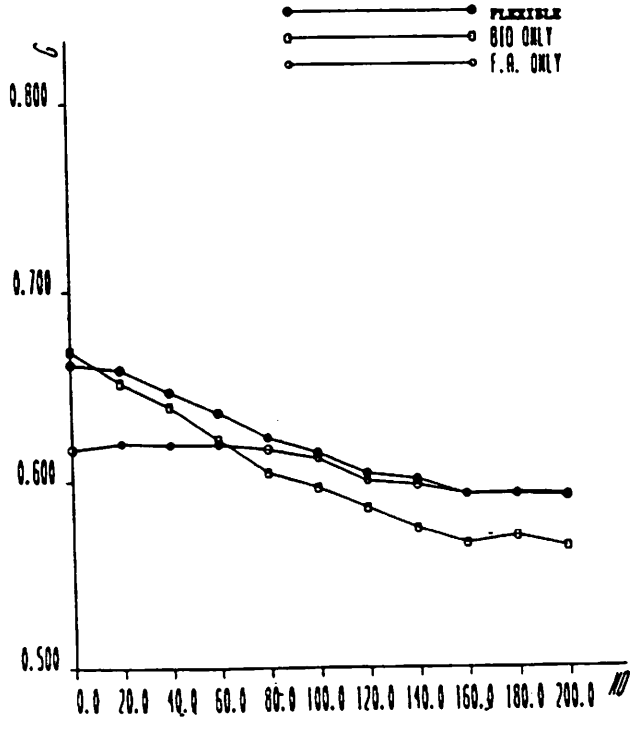


Figure 4: Performance of bidding, focussed addressing and flexible algorithms — Over Load

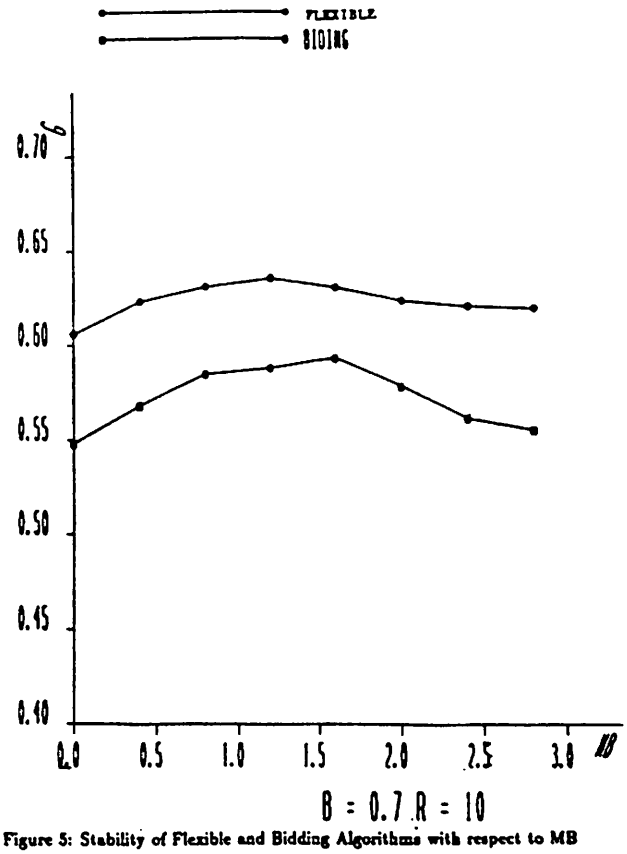


Figure 5: Stability of Flexible and Bidding Algorithms with respect to MB

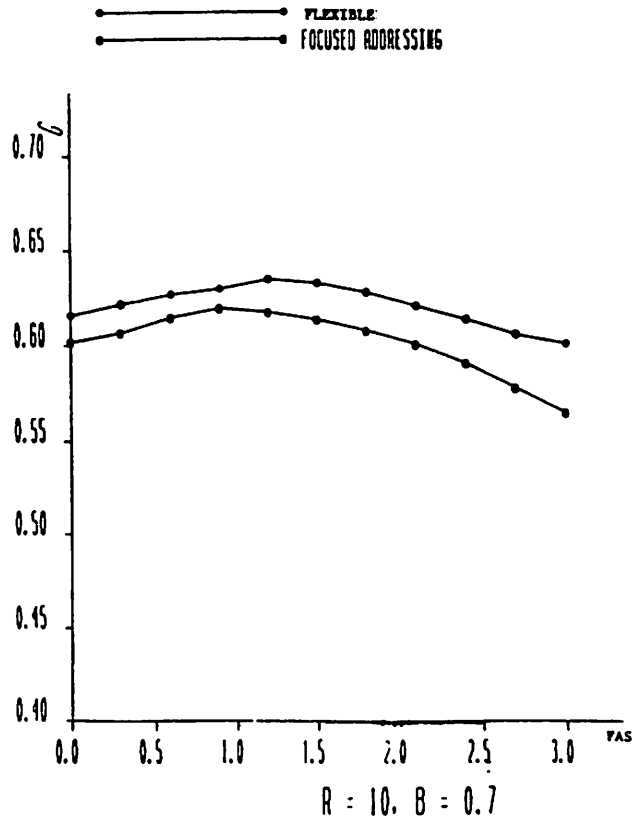


Figure 6: Stability of Flexible and Focussed Addressing Algorithms with respect to FAS

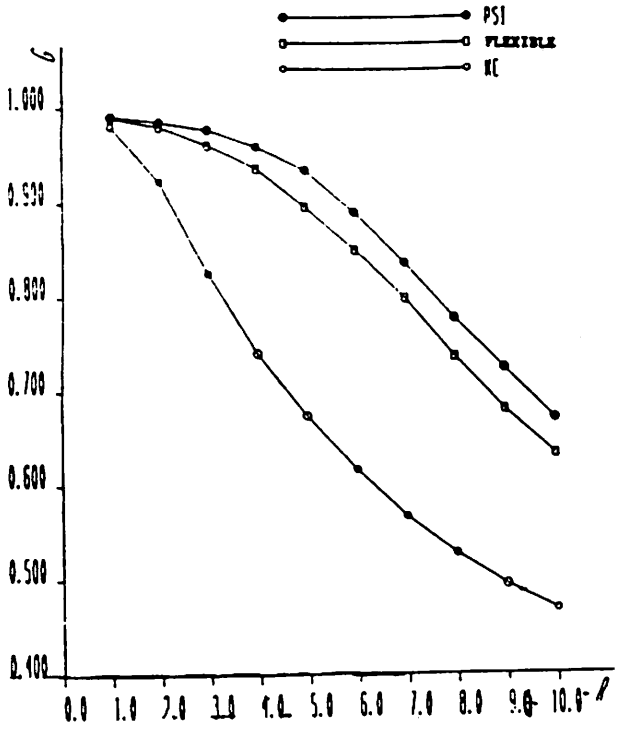


Figure 7: Performance of Non-Cooperative, Perfect-State-Information and Flexible Algorithms — $B = 0.50$

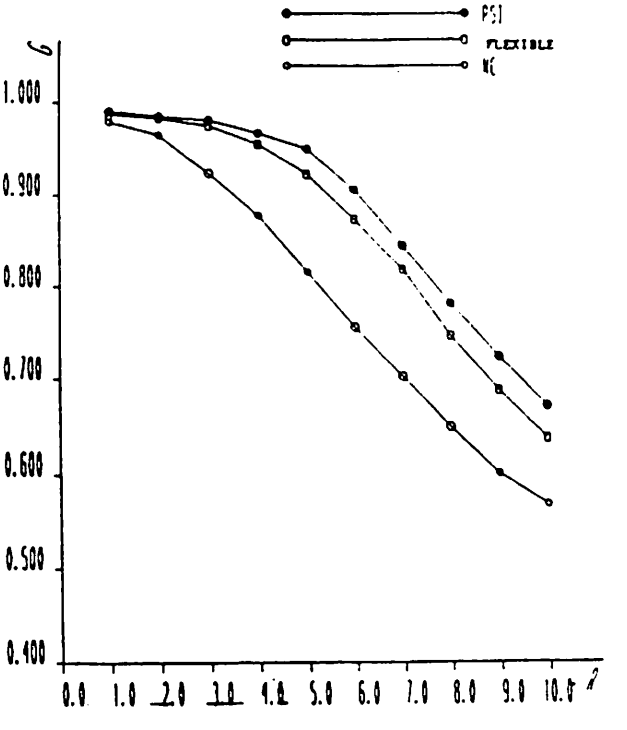


Figure 8: Performance of Non-Cooperative, Perfect-State-Information and Flexible Algorithms — $B = 0.75$

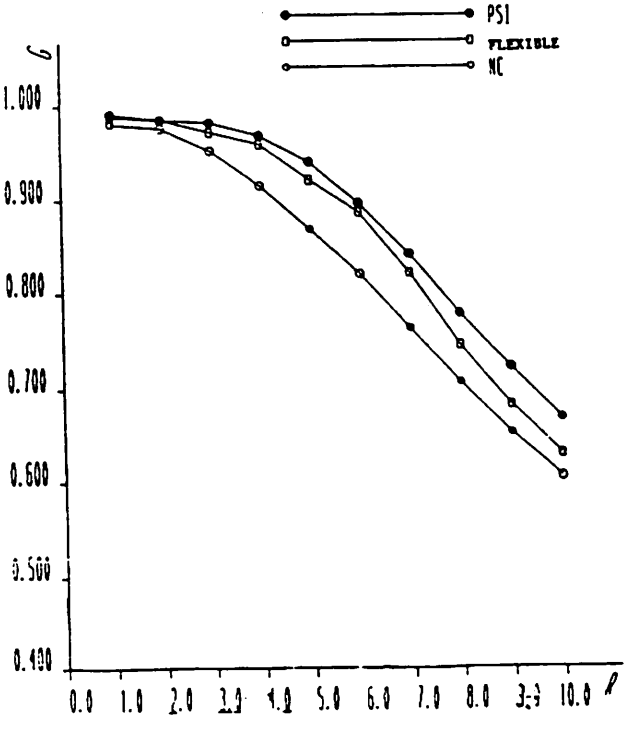
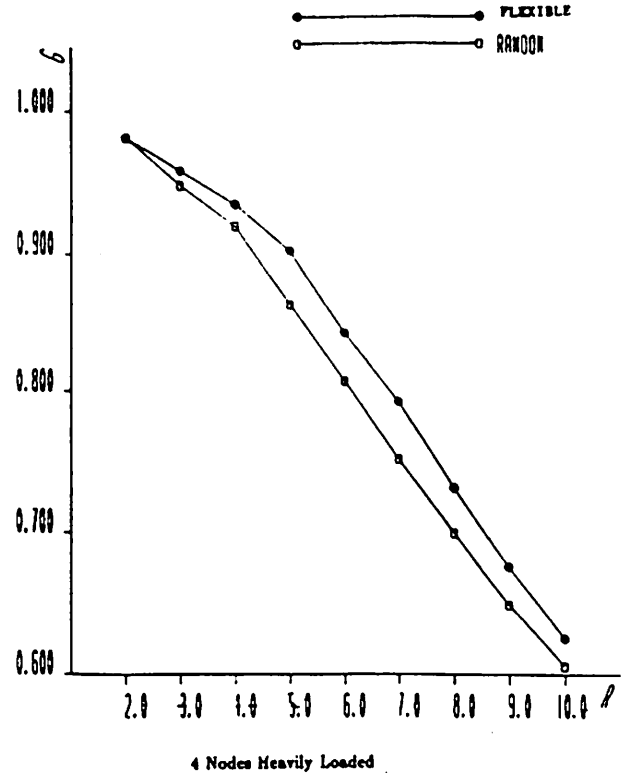
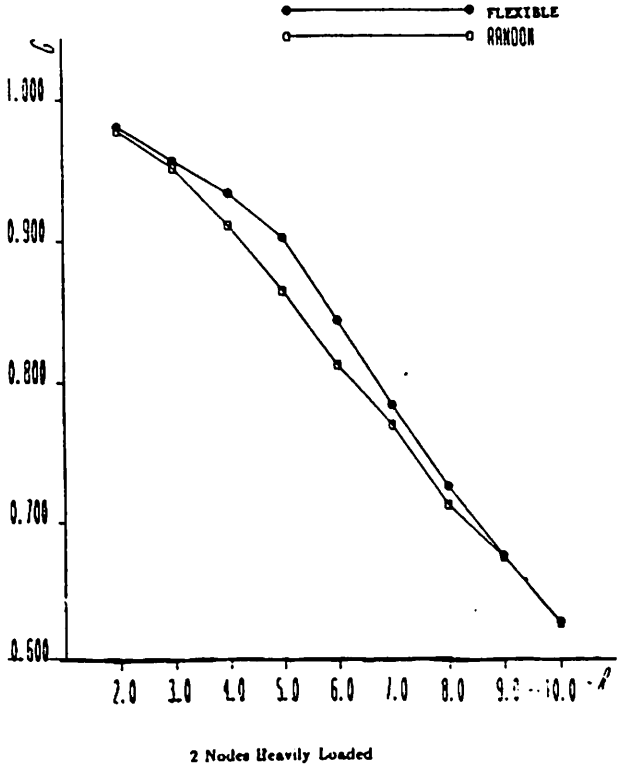


Figure 9: Performance of Non-Cooperative, Perfect State Information and Flexible Algorithms — $B = 1.00$



4 Nodes Heavily Loaded

Figure 10: Performance of Random Scheduling and and Flexible Algorithms -



2 Nodes Heavily Loaded

Figure 11: Performance of Random Scheduling and and Flexible Algorithms

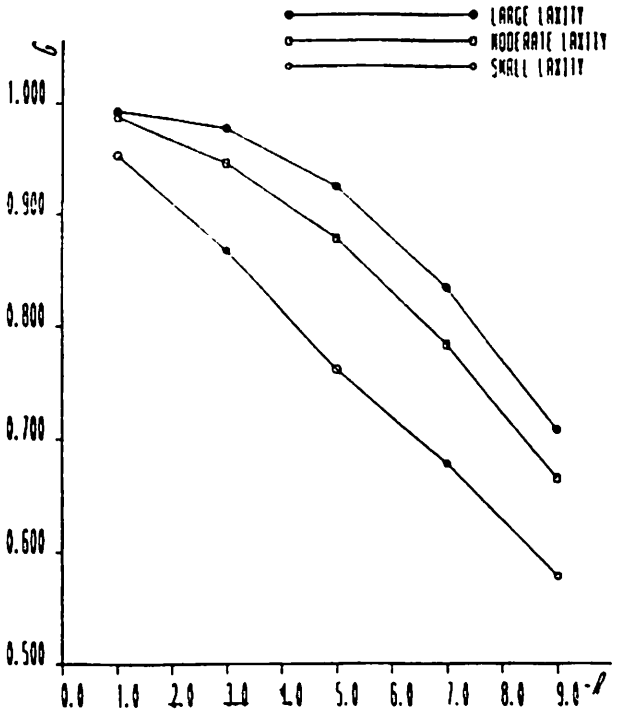


Figure 12: Effect of Task Laxity

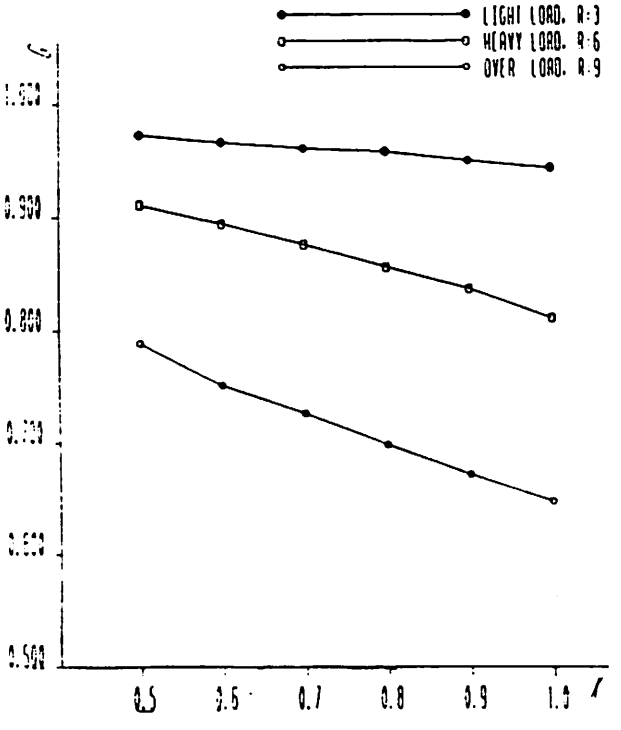


Figure 13: Effect of Reclaiming Unused Task Computation Times

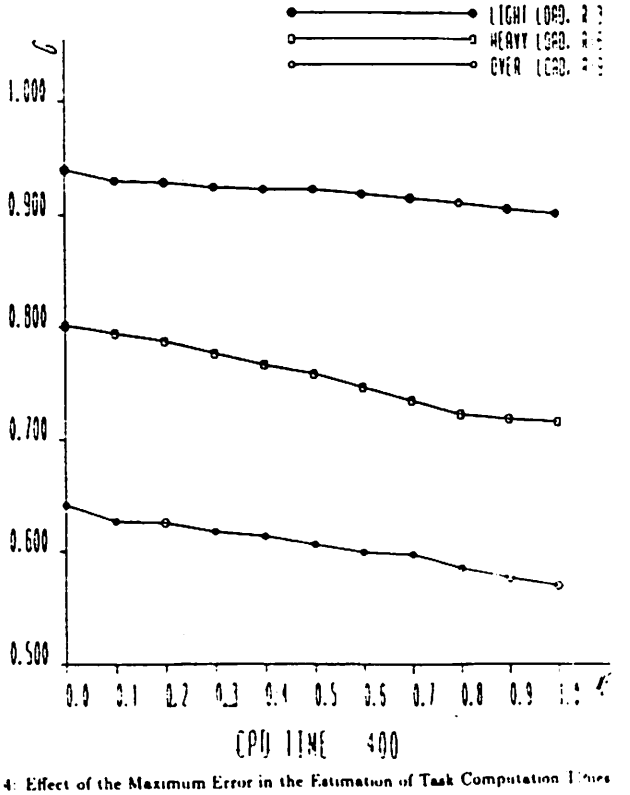


Figure 14: Effect of the Maximum Error in the Estimation of Task Computation Times