SCHEDULING ALGORITHMS FOR HARD REAL-TIME SYSTEMS — A BRIEF SURVEY*[1]

Sheng-Chang Cheng[2]
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598

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

COINS Technical Report 87-55
June 10, 1987

*Preparation of this survey was supported in part by the Office of Naval Research under Grant 048-716/3-22-85 and by the National Science Foundation under Grant DCR-8500332.
[1]This work is part of the *Spring* project conducted at the University of Massachusetts.
[2]This survey was prepared when the first author was at the University of Massachusetts.

ABSTRACT

In hard real-time systems, tasks have to be performed not only correctly, but also in a timely fashion. Otherwise, there might be severe consequences. Typically, a hard real-time task is characterized by its timing constraints, precedence constraints, and resource requirements. In a hard real-time system, task scheduling is the most important problem, because it is the scheduling algorithm that ensures that tasks meet their deadlines. In this paper, we survey proposed solutions for scheduling tasks in hard real-time systems. Our survey includes static and dynamic scheduling algorithms in the context of both distributed and centralized systems. This study of current literature shows that except for a few cases, most of the previous static scheduling approaches are inflexible and cannot be efficiently applied to dynamic scheduling problems. From this study dynamic scheduling emerges as a challenging new problem especially for distributed hard real-time systems.

1 Introduction

A number of new and sophisticated real-time applications are currently being contemplated by governments and industries around the world. Space stations, automated factories of the future, and future command and control systems are examples of such systems. There are two types of real-time systems, namely, *soft* real-time systems and *hard* real-time systems. In soft real-time systems, tasks are performed by the system as fast as possible, but they are not constrained to finish by specific times. On the other hand, in hard real-time systems, tasks have to be performed not only correctly, but also in a timely fashion. Otherwise, there might be severe consequences. Typically, a hard real-time task is characterized by its timing constraints, precedence constraints, and resource requirements. Flight control, automated manufacturing plants, telecommunications, and command and and control systems are examples of such systems. A taxonomy of the scheduling algorithms for real-time systems is depicted in Figure 1. We also examine some approaches to scheduling soft real-time tasks that can be potentially applied to hard real-time systems.

Task scheduling in hard real-time systems can be *static* or *dynamic*. A static approach calculates schedules for tasks off-line and it requires the complete prior knowledge of tasks' characteristics. A dynamic approach determines schedules for tasks on the fly and allows tasks to be dynamically invoked. Although static approaches have low run-time cost, they are inflexible and cannot adapt to a changing environment or to an environment whose behavior is not completely predictable. When new tasks are added to a static system, the schedule for the entire system must be recalculated, which is expensive in terms of time and money. In contrast, dynamic approaches involve higher run-time costs, but, because of the way they are designed, they are flexible and can easily adapt to changes in the environment. In this paper, we survey both static and dynamic scheduling algorithms for hard real-time systems. However, because of the enormous amount of literature which deals with hard real-time scheduling problems, it is impossible to discuss all the material. Therefore, we only present an overview of previous static and dynamic scheduling approaches and discuss their relationship.

A centralized system is one in which the processors are located at a single point in the sys-

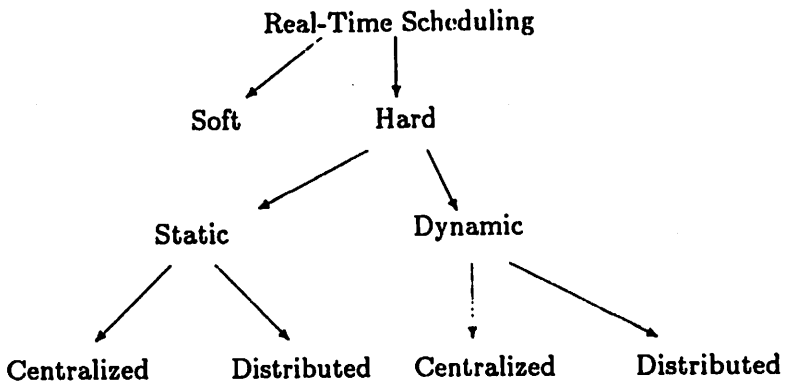

Figure 1: A taxonomy of the real-time scheduling algorithms.

tem and the inter-processor communication cost is negligible compared to the processor execution cost. A multiprocessor system with shared memory is an example of such system. In contrast, a distributed system is one in which the processors are distributed at different points in the system and the inter-processor communication cost is not negligible compared to the processor execution cost. A local area computer network is an example of such system. In distributed systems, inter-processor communication cost is an important factor which must be explicitly taken into account in scheduling.

The remainder of this survey is organized as follows. Section 2 describes possible forms of the scheduling problem in hard real-time systems, in terms of the system model, the nature of tasks to be scheduled, and the objectives of scheduling algorithms. Section 3 surveys static scheduling algorithms for both centralized and distributed systems. Section 4 discusses dynamic scheduling algorithms. Finally, section 5 summarizes this survey.

2 Scheduling Problems

A scheduling problem in a hard real-time system is defined by the model of the system, the nature of tasks to be scheduled, and the objectives of a scheduling algorithm. Each of these is described below.

2.1 System Models

A hard real-time system consists of one or more nodes connected by a communication network. Each node consists of one or more processors along with a set of resources that may be requested by application tasks. The scheduling algorithm determines the order of execution of tasks on each processor and resource in such a way that resource and timing requirements of tasks are met. In many hard real-time systems, resources required by a task are assumed to be available as soon as the task is invoked. For such systems, processors form the primary resources.

2.2 Nature of Tasks

A *task* is a software module that can be invoked to perform a particular function. A task is the scheduling entity in a system. In static systems, the set of tasks to be executed by a system is prespecified for the system, so the number of tasks to be scheduled in a system is known beforehand. However, in dynamic systems, new tasks are allowed to arrive (to be invoked) at unpredictable times so the number of tasks that must be scheduled changes at run-time. In a hard real-time system, a task is characterized by its timing constraints, precedence constraints, as well as resource requirements. Except where explicitly stated otherwise, in this survey, we assume that the resource requirements, except for the processor resources, are always met. [1]

The timing constraints of a task are specified in terms of one or more of the following parameters:

- The arrival time, A: The time at which a task is invoked in the system.
- The ready time, R: The earliest time at which a task can begin execution. The ready time of a task is equal to or greater than its arrival time.
- The worst case computation time, C: The execution time of a task is always less than this amount of time.
- The deadline, D: The time by which a task must finish.

[1] A survey of scheduling algorithms for real-time systems constrained by arbitrary types of resources can be found in [ZRS87c].

In a static system, all the tasks and their timing constraints are known beforehand. In particular, the arrival times of tasks are prespecified. However, in a dynamic system, tasks arrive at arbitrary times so the arrival time of tasks is unpredictable.

In many conventional hard real-time systems, tasks are assigned with fixed *priorities* to reflect *critical* deadlines, and tasks are executed in an order determined by the priorities. During the testing period, the priorities are (usually manually) adjusted until the system implementer is convinced that the system *works* [LTJ85]. Such an approach can only work for relatively simple systems, because it is difficult to determine a *good* priority assignment for a system with a large number of tasks by such a *test-and-adjust* method. Also, such fixed-priority assignment approaches suffer the same problem as other static scheduling approaches. Once the priorities are *fixed* on a system, it is very expensive to modify the priority assignment. Because of these reasons, we do not discuss such approaches in this survey.

In many hard real-time systems, tasks can be *non-periodic* or *periodic*. A nonperiodic task is one which when invoked is expected to execute just once, and has an arbitrary arrival time and deadline. For this survey, a periodic task is defined as one which is invoked exactly once per period P. The arrival time of an instance of a periodic task specifies the time at which the instance of the periodic task is invoked. The arrival time and the deadline of instances of a periodic task with period P are specified as follows:

$$A(i+1) = D(i)$$

$$D(i+1) = A(i+1) + P$$

where $A(i)$ and $D(i)$ are the arrival time and the deadline of the i-th instance of the periodic task, respectively.

The precedence constraints among a set of tasks specify the relations between the tasks. A task T_i is said to *precede* task T_j if T_i must finish before T_j begins. Interrelated tasks communicate with each other in real-time to achieve synchronization as well as to exchange data. The precedence graph of a set of tasks is an acyclic directed graph. A precedence graph may be a chain, a tree, a series-parallel graph, or an arbitrary one. In static systems, the set of tasks to be scheduled and

the precedence graph are known in advance. In dynamic systems, new sets of interrelated tasks dynamically arrive and the precedence graph of a new task set is known only when the task set arrives.

In hard real-time systems, tasks are also distinguished as *preemptable* and *nonpreemptable*. A task is preemptable if its execution can be interrupted by other tasks at any time and resumed afterwards. A task is nonpreemptable if it must run to completion once it starts. Whether a task is preemptable or not is mainly determined by the nature of the application environment.

There are, of course, many other possible types of constraints such as placement constraints. We do not discuss them here.

2.3 Objectives of Scheduling Algorithms

The function of a scheduling algorithm is to determine, for a given set of tasks, whether a schedule (the sequence and the time periods) for executing the tasks exists such that the timing, precedence, and resource constraints of the tasks are satisfied, and to calculate such a schedule if one exists. In static systems, a scheduling algorithm determines the schedule for a set of tasks off-line. However, in dynamic systems, because not all the characteristics of tasks are known *a priori*, a scheduling algorithm determines the schedule for tasks on-line and *progressively*. A scheduling algorithm is said to *guarantee* a newly arriving task if the algorithm can find a schedule for all the previously guaranteed tasks and the new task such that each task finishes by its deadline. If a scheduling algorithm guarantees a task, it ensures that the task finishes by its deadline. A major performance metric for a dynamic scheduling algorithm is the *guarantee ratio*, which is the total number of tasks guaranteed versus the total number of task that arrive.

A static scheduling algorithm is said to be *optimal* if, for any set of tasks, it always produces a schedule which satisfies the constraints of the tasks whenever any other algorithm can do so. A dynamic scheduling algorithm is said to be *optimal* if it always produces a feasible schedule whenever a static scheduling algorithm with complete prior knowledge of all the possible tasks can do so. An optimal dynamic algorithm maximizes the guarantee ratio of tasks that ever arrive in a system. As we shall explain in Section 4.1, for most variations of the scheduling problem, to find

such an optimal solution is difficult and computationally intractable. Therefore, an approximate algorithm is necessary. An approximate algorithm with high guarantee ratio is considered to be better than another with a low guarantee ratio.

3 Static Scheduling Algorithms

In this section, we survey static scheduling algorithms for both centralized and distributed systems.

3.1 Static Scheduling in Centralized Systems

This category of scheduling algorithms include scheduling in multiprocessor systems and machine scheduling as done in operations research. We survey algorithms which can handle tasks with precedence constraints as well as those which consider only mutually independent tasks. In each case, both preemptive scheduling and nonpreemptive scheduling are considered.

3.1.1 Scheduling Mutually Independent Tasks

Preemptive Scheduling If preemption is allowed, it is often possible to find a polynomial-time optimal algorithm for a scheduling problem. For uniprocessor systems, Horn [Hor74] developed an $O(n^2)$ algorithm to schedule tasks with arbitrary ready times and deadlines, where n is the number of tasks to be scheduled. His approach is based on the *earliest-deadline-first* policy: tasks with earlier deadlines and earlier ready times are chosen to run before tasks with later deadlines and later ready times. For multiprocessor systems, Horn described an $O(n^3)$ algorithm to schedule tasks with arbitrary ready times and deadlines. His approach is based on the network flow method and considered only processors with identical processing speed. This approach was recently extended by Martel [Mar82] to consider processors with different speeds. The extended scheduling problem is more difficult and the complexity of Martel's algorithm is $O(m^2n^4 + n^5)$, where m is the number of processors in a system.

The above approaches can also be applied to periodic tasks. For example, for uniprocessor systems, to determine the schedulability for a set of periodic tasks, we only have to consider the instances of the periodic tasks within a time interval between zero and the *least-common-multiple*

of the tasks' periods. Then, we apply the earliest deadline scheme to determine whether all the instances of the periodic tasks in the interval can be scheduled to meet their deadlines. If they can be scheduled, then all the other instances of the periodic tasks can also be scheduled. Likewise, Horn and Martel's approach can also be applied to multiprocessor systems in the same way. However, if the periods of the periodic tasks are relatively prime, these approaches may not be practical, because the number of instances of the periodic tasks to be considered is large and the cost becomes high.

Many researchers have developed efficient scheduling algorithms specifically for periodic tasks. For uniprocessor systems, Liu and Layland [LL73] developed a *rate-monotonic priority* scheme to determine the schedulability of a set of periodic tasks. Their approach assigns higher priorities to tasks with shorter periods. They showed that this scheme is optimal among fixed-priority schemes. Teixeira [Tei78] presented a fixed-priority assignment scheme for a slightly different problem. He assumed that the relative deadline of a periodic task can be different from the period of the task. Sha et. al. [JL86] describe a technique to modify the periods of tasks in such a way that while tasks' timing constraints continue to be met, better processor utilization is achieved. This modification consists of breaking up one periodic task into two, each with half the computation time and half the period as the original task. The above approaches are different from the conventional priority-driven scheduling approaches, because they assign priorities to tasks based on a simple function of the timing constraints, instead of one that combines timing constraints and *criticalness*, of the periodic tasks.

Scheduling periodic tasks on multiprocessor systems is more complicated. Many researchers adopted a partition approach to solve this problem. The main idea of these approaches is to partition a set of periodic tasks among a minimum number of processors such that each partition of the periodic tasks can be scheduled on one processor according to the earliest deadline scheme or the rate-monotonic priority scheme. Davari and Dhall [DD86] showed that, if the earliest deadline scheme is used, a *bin-packing* algorithm can be used to determine a suboptimal partition pattern of periodic tasks among multiple processors. Bannister and Trivedi [BT83] proposed a simple *best-fit* partition scheme. Their approach can be used in conjunction with both the earliest deadline

scheme and the rate-monotonic priority scheme. For rate-monotonic priority scheme, Dhall and Liu [DL78] developed a *next-fit* scheme and a *first-fit* partition scheme. Recently, Davari and Dhall [DD86] improved these schemes and developed a more efficient *next-fit* partition scheme. The time complexity of this improved scheme is $O(n)$.

As described above, many of the scheduling algorithms designed for periodic tasks are based on a fixed-priority assignment scheme. The advantage of a fixed-priority assignment scheme is that they have very small scheduling overheads, because they are designed for prioritized-interrupt handling systems and the priority mechanism is often supported by hardware. However, as we explained earlier, in general, these schemes are very inflexible, because it is expensive to change the priority assignment once it is *fixed* on a system.

Nonpreemptive Scheduling Nonpreemptive scheduling is more difficult than preemptive scheduling. Many nonpreemptive scheduling problems have been shown to be *NP-hard*. For example, scheduling nonpreemptable tasks with arbitrary ready times is *NP-hard* even in uniprocessor systems [LRB77]. For multiprocessor systems, a nonpreemptive scheduling problem is *NP-hard* even when the ready times and deadlines of tasks are the same [Ull76]. Much work has been done on more restrictive problems for which efficient algorithms are available. For uniprocessor systems, Moore [Moo68] showed that the earliest deadline algorithm is optimal for scheduling a set of tasks with the same ready time. Kise [Kis78] developed an $O(n^2)$ algorithm for the case in which a task has an earlier ready time if and only if it has an earlier deadline. For multiprocessor systems, a polynomial optimal algorithm is available only for scheduling tasks with unit computation time [Sim80,Sim83,SS84,LF76].

Many researchers have attempted to solve the general case of the scheduling problem by developing an efficient enumeration algorithm with strong bounding conditions. For uniprocessor systems, Bratley, Florian, and Robillard [BFR71] developed an implicit enumeration algorithm to determine schedule for tasks with arbitrary ready times and deadlines [BFR75]. Baker and Su [BS74] used a similar approach to minimize the maximum tardiness of tasks. Erschler *et al* [EFMR83] developed a necessary condition for scheduling tasks with arbitrary ready times and deadlines. Their theories

can be used to reduce the search space of an enumeration algorithm. For multiprocessor systems, Bratley, Florian, and Robillard [BFR75] developed a multi-stage enumeration algorithm to schedule tasks with arbitrary ready times and deadlines. Because the worst case cost is exponential, the above approaches are designed to run off-line.

Recently, Blazewicz, Drabowski, and Weglarz [BDW86] investigated an interesting scheduling problem in which tasks need multiple processors at the same time for processing. They showed that polynomial-time algorithms exist if the number of processors and the processing times required by tasks are constant. They presented two such algorithms which run in $O(n)$ time. They also showed that, for preemptable tasks with arbitrary computation times, a linear algorithm exists if the tasks require either one or a fixed number of processors.

3.1.2 Scheduling Tasks with Precedence Constraints

For uniprocessor systems, most scheduling problems which consider precedence constraints can be solved in polynomial time. Lawler [Law73] showed that scheduling nonpreemptable tasks with deadlines and arbitrary precedence constraints can be solved by the *latest deadline first algorithm* in $O(n^2)$ time. Each task has a deadline. Tasks are scheduled from last to first one at a time. Each time, the task with the latest deadline is chosen from among those whose successors have been scheduled. Blazewicz [Bla76] proved that, for this scheduling problem, a preemptive schedule exists if and only if a nonpreemptive schedule exists. Therefore, in this case, preemption need not be considered. He also showed that the earliest deadline algorithm can be used to schedule preemptable tasks with arbitrary ready times and precedence constraints. The main idea in his approach is to modify the ready times and deadlines of tasks such that they comply with the precedence constraints of the tasks. Therefore, precedence constraints need not be explicitly considered.

Scheduling tasks with arbitrary precedence constraints in multiprocessor systems is a much more difficult problem than in uniprocessor systems. For example, scheduling tasks with arbritrary precedence constraints and unit computation time is *NP-hard* both for the preemptive and the nonpreemptive cases [Ull75,Ull76].

Many researchers have attempted to develop efficient heuristic algorithms to solve the general

case of the scheduling problem. For example, Kasahara and Narita [KN84] have developed an implicit heuristic search algorithm to determine the minimum schedule length for a set of non-preemptable tasks with arbitrary precedence constraints. They showed that their enumeration algorithm can provide optimal or suboptimal solutions to large-scale problems within a time limit. However, because the worst case execution time grows exponentially, their algorithm is practical only for static scheduling problems. Elsayed [Els82] presented a number of heuristic algorithms for finding suboptimal solutions to a similar scheduling problem. These heuristic algorithms do not enumerate over multiple paths in a search space. They are designed based on a straightforward topological search scheme and the *critical path* method combined with a heuristic rule. Therefore, such heuristic algorithms are much more efficient than the implicit enumeration algorithm described above. These algorithms can be applied to dynamic scheduling problems.

For tasks whose precedence graph is a tree, several efficient algorithms have been found. Chandy and Reynolds [CR75] showed that, for two-processor systems, the *highest-level-first* policy is optimal for nonpreemptive scheduling. The *level* of a task in a tree is defined to be the length of the longest path between the task and any *leaf* task in the tree. The complexity of this policy is $O(n)$. This policy attempts to choose to run, among the remaining tasks that are ready for processing, any one of those that are at the highest level in a tree. However, they also showed that this policy is not optimal for three-processor systems. Hu [Hu61] developed an $O(n \log n)$ algorithm to schedule nonpreemptable tasks with unit computation time. For preemptable tasks, two efficient optimal algorithms that can handle tasks with arbitrary computation times have been found. Muntz and Coffman [MC70] developed a preemptive scheduling algorithm to calculate the minimum schedule length for tasks in a *reverse* precedence tree (one with its root at the bottom). Their approach attempts to assign processing power to tasks from root to leaves in a bottom-up fashion. Unfinished tasks at levels closer to the leaves are assigned with higher share of processing power; tasks at the same level are assigned with the same share. Their algorithm requires $O(n^2)$ time. Gonzalez and Johnson [GJ77] considered regular trees and their approach attempts to assign processing power to tasks from root to leaves in a top-down fashion. The shares of processing power assigned to tasks are proportional to the total remaining processing times in the subtrees rooted at the tasks.

Their algorithm runs in $O(n \log m)$ time. The above algorithms are designed only for tree-type precedence graphs and for centralized and static systems.

Manacher [Man67] studies an interesting anomaly found in scheduling tasks with precedence constraints in multiprocessor systems. When a set of tasks are scheduled to meet their deadlines according to maximum computation time, they may miss deadlines at run time if the actual computation time is less than the worst case time. This anomaly is due to parallel tasks being executed in an order different from the original schedule. He proposed an algorithm to solve this problem by imposing additional precedence constraints on tasks to preserve the order of tasks which can run in parallel. The above anomaly is not applicable to a distributed system if the system does not maintain a central queue of tasks, since tasks scheduled at different nodes are executed only by those nodes.

3.2 Static Scheduling in Distributed Systems

This category of scheduling problems have been traditionally formulated as *task allocation problems*. In a traditional distributed system, the objective of a task allocation algorithm is to assign tasks to processors in the system so as to balance the loads of processors and to minimize the communication cost for tasks. Task allocation is a difficult problem even without timing constraints. For example, finding optimal assignment of tasks with an arbitrary communication graph to four or more processors with different speeds is known to be *NP-hard* [BS81]. Considerable efforts have been spent on more restrictive allocation problems or on developing heuristic algorithms to find suboptimal solutions. Most of the results can be extended and applied to (static) hard real-time systems to find suboptimal solutions.

Stone [Sto77a,Sto77b] developed network flow algorithms to allocate tasks with arbitrary communication patterns in dual-processor and three-processor systems. Bokhari [BS81] developed a dynamic programming algorithm to allocate tasks which form a tree to a system with an arbitrary number of processors. The time complexity of his algorithm is $O(nm^2)$. His approach was recently extended by Towsley [Tow86] to handle tasks with series-parallel precedence graphs. This algorithm runs in $O(nm^3)$ time. The above approaches consider heterogeneous processors. However, these

approaches attempt to minimize the total execution cost and the communication cost of tasks. They do not attempt to balance load of the processors.

Even though the above approaches are not designed for hard real-time scheduling, they can be extended and applied to hard real-time systems in the following way. The results obtained by the above approaches are a task-to-processor allocation pattern which meets certain optimization criteria without specifing the order of execution of tasks on a processor. We can use such an allocation pattern, in conjunction with heuristic rules for ordering tasks allocated to each processor, to determine a schedule length for tasks and determine whether the schedule meets the timing constraints. Tasks allocated to a processor can be ordered by a heuristic rule such as one that is based on the length of the critical path for a task. After the order of tasks is determined, we can calculate the the schedule length of tasks by a depth-first search algorithm in $O(n)$ time.

Efe [Efe82] developed a heuristic allocation algorithm to balance processor load and to minimize communication cost. His algorithm consists of two phases. First, tasks are clustered with each other to optimize the communication cost and each cluster of tasks is assigned to a processor. Then, tasks are shifted from overloaded to underloaded processors in order to meet load-balance constraints. The algorithm is repeated until a satisfactory degree of load-balancing is achieved. Lo [Lo84] developed heuristic algorithms which incorporate a cost function to maximize the concurrent execution of tasks, in addition to minimizing the total execution and communication costs. These approaches can also be applied to hard real-time systems in the same way as described above.

Many researchers have developed integer programming models to find optimal allocation for tasks with explicit timing constraints. For example, Ma *et al* [MLT82] developed such a model and applied a branch and bound algorithm to solve the allocation model. They showed that their approach can be used to balance processor loads and to minimize communication cost with an example of air defense application. The precedence graph of the tasks in the application is represented by an *AND-OR* graph and each thread of tasks is constrained by a *port-to-port* time. The advantage of such models is that sophisticated constraints, such as allocation preference, task exclusion, redundancy, time, and resource constraints can be incorporated into a model to describe an allocation problem. However, the disadvantage is its exponential computational cost. As a

result, such approaches can not be used on-line.

Leinbaugh and Yamini [LY82] developed an analysis algorithm to compute the worst case finish time for a set of tasks which run on a dedicated network. Each task consists of multiple segments with series-parallel precedence constraints. By assuming that each task is blocked by every other possible task due to both resource sharing and inter-task communication, they compute the maximum blockage time of each task. Their result can also be applied to hard real-time systems. However, their approach does not attempts to balance load of nodes and to minimize communication cost.

Recently, Peng and Shin [PS87] developed a generalized stochastic Petri net to model the behavior of a distributed hard real-time system. They considered periodic tasks with precedence constraints and nonperiodic tasks without predecence constraints. Tasks are statically allocated to nodes in the system. The precedence constraints of tasks form an *AND-OR* graph. They used the Petri net to build a sequence of homogeneous continuous-time Markov chains to model the concurrent task execution in the system. Given a task selection policy and the local state of each node in the system, the Markov-chain model can be used to compute the probability of missing a hard deadline.

4 Dynamic Scheduling Algorithms

In this section, we survey dynamic scheduling algorithms for both centralized systems and distributed systems.

4.1 Dynamic Scheduling in Centralized Systems

Theoretically, all the static scheduling algorithm for centralized systems can run on-line to guarantee tasks. But, except for a few cases, most of the algorithms which are optimal for static scheduling are not optimal for dynamic scheduling. In particular, Mok and Dertouzos [MD78] showed that, for multiprocessor systems, there can be no optimal algorithm for scheduling preemptable tasks if the arrival time of tasks are not known *a priori*. Furthermore, because run-time cost is an important factor for dynamic scheduling, most sophisticated static algorithms, including many multi-stage

polynomial algorithms, are not appropriate for dynamic scheduling. Because of these reasons, heuristic algorithms become important to dynamic scheduling problems.

For uniprocessor systems, Dertouzos [Der74] showed that the earliest deadline algorithm is optimal for scheduling preemptable tasks with arbitrary arrival times. However, the cost of invoking the algorithm to guarantee tasks at run-time is not addressed. Ramamritham and Stankovic [RS84] described a guarantee scheme which is based on the earliest deadline policy but which takes into account run-time cost. For scheduling nonpreemptable tasks, Baker and Su [BS74] compared four simple heuristic algorithms which schedule tasks according to an order determined by ready time, by deadline, by the average of the ready time and deadline, and by both the ready time and the deadline, respectively. In a limited number of tests, they showed that the last two algorithms perform better than the first two.

For scheduling preemptable tasks on multiprocessor systems, Mok and Dertouzos [MD78] showed that, if the set of all possible tasks that will ever arrive in a system can be scheduled initially, then the set of tasks can also be scheduled at run-time. They have also proved that one successful run-time algorithm is the least laxity algorithm. Obviously, the use of this approach is limited, because in most dynamic systems, the probability that all possible arriving tasks can be scheduled initially is low. Recently, Locke, Tokuda, and Jensen [LTJ85], compared a number of simple dynamic scheduling policies. They found that the *least laxity first* and the *earliest deadline first* are two good heuristic policies. Zhao, Ramamritham, and Stankovic [ZRS87b] developed a heuristic function and an efficient backtracking scheme for scheduling nonpreemptable tasks with resource constraints. They found that with a limited number of backtracking, the success ratio of their search algorithm for scheduling tasks can be as high as 99.5% of that of an exhaustive search algorithm. Their approach was recently extended to preemptable tasks [ZRS87a].

4.2 Dynamic Scheduling in Distributed Systems

A dynamic scheduling algorithm for a distributed system should maximize the guarantee ratio of tasks in the network-wide system. This is a much more complicated problem than that in a centralized system. In a distributed system, tasks may dynamically arrive at each node in the

system. Therefore, the status of each node is changing constantly and cannot be decided in advance. A scheduling algorithm which attempts to work for the good of the entire system will require state information from more than one node, but that information can only be acquired at run time. Furthermore, because of the communication delay, no state information of remote nodes is accurate. A scheduling algorithm must make decisions based on out-of-date state information. Because of these complications, dynamic scheduling algorithms for distributed systems often consist of two components, namely, a a *local scheduling* algorithm and a *distributed scheduling* algorithm. A local scheduling algorithm dynamically decides whether tasks which arrived at a node can be scheduled on that node. A distributed scheduling algorithm dynamically decides where in the network a task that cannot be scheduled at a node should be transferred to be scheduled. Dynamic scheduling algorithms for centralized systems can be used for local scheduling in distributed systems. However, distributed scheduling is a new problem for hard real-time systems. Our following discussion focuses on distributed scheduling algorithms.

Many distributed scheduling algorithms have been proposed for traditional distributed systems. The objective of these algorithm is to balance loads among nodes in a system. Thus, these algorithms are referred as *load-balancing* algorithms in literature [Cas81,ELZ86,LM82], [Sta84,Sta85a,Sta85b,SM86]. Most of the load balancing algorithms do not consider timing constraints of tasks and cannot be directly applied to hard real-time systems. Chang and Livny [CL86] have developed and evaluated scheduling algorithms that are aimed at reducing the tardiness of tasks. Ramamritham and Stankovic [RS84] developed a heuristic algorithm for scheduling mutually independent tasks in distributed hard real-time systems. Their goal is to schedule tasks such that as many tasks as possible can be *guaranteed* to complete before their deadlines. Their approach consists of a *bidding* algorithm and a *focussed addressing* algorithm. In focussed addressing, if a task cannot be guaranteed at a node, it is sent to a selected remote node that is estimated to have high surplus processing time. In bidding, a task is sent to a remote node selected based on the bids that nodes send for the task. Scheduling decision made in focussed addressing is based on inaccurate state information, but it entails low communication delay. The communication delay involved in bidding is high, but the selection is based on relatively accurate state information of nodes.

They showed how to combine these two algorithms in a distributed hard real-time system and how to take into account the timing constraints of tasks in the algorithm. Simulation results of this algorithm are reported in [SRC85,RSZ86]. These results show that dynamic and distributed hard real-time scheduling is feasible and a system can benefit substantially from distributed scheduling under a wide range of system conditions and task parameters.

Precedence constraints of tasks add one additional dimension to dynamic scheduling problems of distributed systems. It is a difficult problem to dynamcially decide how to distribute a group of tasks with precedence constraints in a network. Most of the static scheduling algorithms which consider precedence constraints cannot be used in dynamic systems, because these algorithms require complete knowledge of tasks in the entire system and tend to search for an allocation pattern among the complete set of tasks. This is not practical in a dynamic system. Cheng, Stankovic, and Ramamritham [CSR86] developed a novel algorithm to solve this problem when each group, has a deadline. For a group that must be distributed, their approach attempts to partition tasks in the group into subgroups and distribute the subgroups in the network to be scheduled in parallel. Tasks in a group are scheduled to run either completely or not at all. The algorithm that combines focussed addressing and bidding is used to determine how to distribute the subgroups in the network. Their approach provides a solution to a combination of several problems, i.e., their solution is for dynamic, distributed, hard real-time systems where tasks have arbitrary precedence constraints. The simulation results of this approach are reported in [CSR87,Che87]. The results showed that dynamic scheduling is practical under a wide range of system conditions and for groups of different characteristics, i.e., different sizes, different precedence constraints, and different timing constraints. Also, their results showed that a system can benefit substantially from distributed scheduling.

5 Summary

We have presented a sample of previous scheduling algorithms for hard real-time systems. We first defined the possible forms of the scheduling problems for such systems. Then, we discussed the previous approaches to both static and dynamic scheduling problems. In each case, we studied

both centralized and distributed systems. The survey has shown that, except for a few cases, most of the previous static scheduling approaches are inflexible and cannot be efficiently applied to the dynamic scheduling problems. The survey also showed that dynamic scheduling is a challenging new problem to distributed hard real-time systems. In dynamic distributed systems, it is necessary for nodes to interact with each other to determine how to distribute tasks across the network to meet time constraints and to adapt the decisions to the current state of the network. Based on the results reported in current literature, dynamic scheduling appears to be feasible for distributed systems under a wide range of system conditions and for tasks of different characteristics.

References

[BDW86] J. Blazewicz, M. Drabowski, and J. Weglarz. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Trans. on Computer*, C-35(5), 1986.

[BFR71] P. Bratley, M. Florian, and P. Robillard. Scheduling with earliest start and due date constraints. *Naval Research Logistics Quarterly*, 18(4), December 1971.

[BFR75] P. Bratley, M. Florian, and P. Robillard. Scheduling with earliest start and due date constraints on multiple machines. *Naval Research Logistics Quarterly*, 22, 1975.

[Bla76] J. Blazewicz. Scheduling dependent tasks with different arrival times to meet deadlines. In E. Gelenbe, editor, *Modelling and Performance Evaluation of Computer Systems*, North-Holland, 1976.

[BS74] K. R. Baker and Z.-S. Su. Sequencing with due-dates and early start times to minimize maximum tardiness. *Naval Research Logistics Quarterly*, 21, 1974.

[BS81] S. H. Bokhari and H. Shahid. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Trans. on Software Engineering*, SE-7(6), 1981.

[BT83] J. A. Bannister and K. S. Trivedi. Task allocation in fault-tolerant distributed systems. In *Acta Informatica*, Springer-Verlag, 1983.

[Cas81] L. M. Casey. Decentralized scheduling. *The Australian Computer Journal*, 13(2), 1981.

[Che87] S. Cheng. *Dynamic Scheduling Algorithms for Distributed Hard Real Time Systems.* PhD thesis, University of Massachusetts, Amherst, 1987.

[CL86] H.Y. Chang and M. Livny. Distributed scheduling under deadline constraints: a comparison of sender-initiated and receiver-initiated approaches. In *Proceedings of the Real-Time Systems Symposium*, 1986.

[CR75] K. M. Chandy and P. F. Reynolds. Scheduling partially ordered tasks with probabilistic execution times. In *Proceedings of the Fifth Symposium on Operating Systems Principles*, 1975.

[CSR86] S. Cheng, J. A. Stankovic, and K. Ramamritham. Dynamic scheduling of groups of tasks with precedence constraints in distributed hard real-time systems. In *IEEE Real-Time System Symposium*, 1986.

[CSR87] S. Cheng, J. A. Stankovic, and K. Ramamritham. Dynamic scheduling of groups of tasks with precedence constraints in distributed hard real-time systems. 1987. To be submitted to *IEEE Trans. of Computers.*

[DD86] S. Davari and S. K. Dhall. An on line algorithm for real-time tasks allocation. In *IEEE Real-Time Systems Symposium*, December 1986.

[Der74] M. Dertouzos. Control robotics: the procedural control of physical processes. In *Proc. of the IFIP Congress*, 1974.

[DL78] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1), 1978.

[Efe82] K. Efe. Heuristic models of task assignment scheduling in distributed systems. *IEEE Computer*, June 1982.

[EFMR83] J. Erschler, G. Fontan, C. Merce, and F. Roubellat. A new dominance concept in scheduling n jobs on a single machine with ready times and due dates. *Operations Research*, 31(1), 1983.

[Els82] E. A. Elsayed. Algorithms for project scheduling with resource constraints. *International Journal of Production Research*, 20(1), 1982.

[ELZ86] D. L. Eager, E. D. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. on Software Engineering*, SE-12(5), 1986.

[GJ77] T. Gonzalez and D. B. Johnson. *A new algorithm for preemptive scheduling of trees.* Technical Report 222, Computer Science Department, Pennsylvania State University, 1977.

[Hor74] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21, 1974.

[Hu61] T. C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9, 1961.

[JL86] J. P. Lehoczky and L. Sha. Performance of bus scheduling algorithms. In *Performance 86*, 1986.

[Kis78] H. Kise. A solvable case of the one-machine scheduling problem with ready and due times. *Operations Research*, 26(1), 1978.

[KN84] H. Kasahara and S. Narita. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. on Computer*, C-33(11), 1984.

[Law73] E. L. Lawler. Optimal scheduling of a single machine subject to precedence constraints. *Management Science*, 19, 1973.

[LF76] T. Lang and E. B. Fernandez. Scheduling of unit-length independent tasks with execution constraints. *Information Processing Letters*, 4(4), 1976.

[LL73] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, 20(1), 1973.

[LM82] M. Livny and M. Melman. Load balancing in homogeneous broadcast distributed systems. In *Proc. ACM Computer Network Performance Symposium*, 1982.

[Lo84] V. M. Lo. Heuristic algorithms for task assignment in distributed systems. In *Proc. Int'l Conf. Distributed Computing Systems*, 1984.

[LRB77] J. K Lenstra, A. H. G. Rinnooy Kan, and P. Bruchker. Complexity of machine scheduling problems. In *Annals of Discrete Mathematics*, North–Holland, 1977. 1.

[LTJ85] C. D. Locke, H. Tokuda, and E. D. Jensen. *A time-driven scheduling model for real-time operating systems.* Technical Report, Carnegie–Mellon University, 1985.

[LY82] D. W. Leinbaugh and M. R. Yamini. Guaranteed response times in a distributed hard real–time environment. In *Proc. IEEE Real-Time Systems Symp.*, December 1982.

[Man67] G. K. Manacher. Production and stablization of real-time task schedules. *J. ACM*, 14(3), 1967.

[Mar82] C. Martel. Preemptive scheduling with release times, deadlines, and due times. *J. ACM*, 29(3), 1982.

[MC70] R. R. Muntz and E. G. Coffman. Preemptive scheduling of real-time tasks on multiprocesssor systems. *J. ACM*, 17(2), April 1970.

[MD78] A. K. Mok and M. L. Dertouzos. Multiprocessor scheduling in a hard real time environment. In *Proc. of the Seventh Texas Conference on Computing Systems*. November 1978.

[MLT82] P.-Y. R. Ma, E. Y. S. Lee, and M. Tsuchiya. A task allocation model for distributed computing systems. *IEEE Trans. on Computer*, C-31(1), 1982.

[Moo68] J. M. Moore. An n job, one machine sequencing algorithm for minimize the number of late jobs. *Management Science*, 15(1), 1968.

[PS87] D. Peng and K. G. Shin. Modeling of concurrent task execution in a distributed system for real-time control. *IEEE Trans. on Computer*, C-36(4), 1987.

[RS84] K. Ramamritham and J. A. Stankovic. Dynamic task scheduling in distributed hard real-time systems. *IEEE Software*, 1(3), 1984.

[RSZ86] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of hard real-time tasks under resource constraints. 1986. submitted for publication.

[RSZ87] K. Ramamritham, J. A. Stankovic, and W. Zhao. Meta-level control in distributed real-time systems. 1987. *International Conference on Distributed Computing Systems.*

[Sim80] B. Simons. A fast algorithm for multiprocessor scheduling. In *Proc. 21st Annual Symposium on Foundation of Computer Science*, 1980.

[Sim83] B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal for Computing*, 12(2), 1983.

[SM86] J. A. Stankovic and Ravi Mirchandaney. Using stochastic learning automata for job scheduling in distributed processing systems. *Journal of Parallel and Distributed Computing*, December 1986.

[SRC85] J. A. Stankovic, K. Ramamritham, and S. Cheng. Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. *IEEE Trans. on Computer*, C-34(12), 1985.

[SS84] B. Simons and M. Sipser. On scheduling unit-length jobs with multiple release time/deadline intervals. *Operations Research*, 32(1), 1984.

[Sta84] J. A. Stankovic. Simulation of three adaptive, decentralized controlled, job scheduling algorithms. *Computer Network*, 8(3), 1984.

[Sta85a] J. A. Stankovic. An application of bayesian decision theory to decentralized control of job scheduling. *IEEE Trans. on Computer*, C-34(2), 1985.

[Sta85b] J. A. Stankovic. Stability and distributed scheduling algorithms. *IEEE Trans. on Software Engineering*, SE-11(10), 1985.

[Sto77a] H. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. on Software Engineering*, SE-3(1), 1977.

[Sto77b] H. Stone. *Program assignment in three-processor systrems and tricutset partitioning of graphs.* Technical Report ECE-CS-77-7, Univeristy of Massachusetts, Amherst, 1977.

[Tei78] T. Teixeira. Static priority interrupt scheduling. In *Proc. of the Seventh Texas Conference on Computing Systems*, November 1978.

[Tow86] D. Towsley. Allocating programs containing branches and loops within a multiple processor system. *IEEE Trans. on Software Engineering*, SE-12(10), 1986.

[Ull75] J. D. Ullman. *np*-complete scheduling problem. *Journal of Computer and System Sciences*, 10, 1975.

[Ull76] J. D. Ullman. Complexity of sequence problems. In E. G. Coffman, editor, *Computer and Job-Shop Scheduling Theory*, J. Wiley, New York, 1976.

[ZR87] W. Zhao and K. Ramamritham. Simple and integrated heuristic algorithms for scheduling tasks with time and resource constraints. 1987. To appear in *J. of Systems and Software.*

[ZRS87a] W. Zhao, K. Ramamritham, and J. A. Stankovic. Preemptive scheduling under time and resource constraints. August 1987. To appear in *IEEE Trans. on Computers.*

[ZRS87b] W. Zhao, K. Ramamritham, and J. A. Stankovic. Scheduling tasks with resource requirements in hard real-time systems. *IEEE Trans. on Software Engineering*, SE-12(5), 1987.

[ZRS87c] W. Zhao, K. Ramamritham, and J. A. Stankovic. A survey of scheduling algorithms for resource constrained real-time systems. February 1987. Technical Report, University of Massachusetts.