Real-Time Systems, 23, 239–271, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

Robust Adaptive Metrics for Deadline Assignment in Distributed Hard Real-Time Systems

JAN JONSSON* janjo@ce.chalmers.se
Department of Computer Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden

KANG G. SHIN kgshin@eecs.umich.edu
Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122, USA

Abstract. Distributed real-time applications usually consist of several component tasks and must be completed by its end-to-end (E-T-E) deadline. As long as the E-T-E deadline of an application is met, the strategy used for dividing it up for component tasks does not affect the application itself. One would therefore like to ''slice'' each application E-T-E deadline and assign the slices to component tasks so as to maximize the schedulability of the component tasks, and hence the application. Distribution of the E-T-E deadline over component tasks is a difficult and important problem since there exists a circular dependency between deadline distribution and task assignment. We propose a new deadline-distribution scheme which has two major improvements over the best scheme known to date. It can distribute task deadlines prior to task assignment and relies on new adaptive metrics that yield significantly better performance in the presence of high resource contention. The deadline-distribution problem is formulated for distributed hard real-time systems with relaxed locality constraints, where schedulability analysis must be performed at pre-run-time, and only a subset of the tasks are constrained by pre-assignment to specific processors. Although it is applicable to any scheduling policy, the proposed deadline-distribution scheme is evaluated for a non-preemptive, time-driven scheduling policy. Using extensive simulations, we show that the proposed adaptive metrics deliver much better performance (in terms of success ratio and maximum task lateness) than their non-adaptive counterparts. In particular, the simulation results indicate that, for small systems, the adaptive metrics can improve the success ratio by as much as an order of magnitude. Moreover, the new adaptive metrics are found to exhibit very robust performance over a large variety of application and architecture scenarios.

Keywords: distribution of an end-to-end deadline, precedence constraints, relaxed locality, hard-real-time systems, non-preemptive scheduling, multiprocessor systems

1. Introduction

A distributed real-time system usually executes a set of applications each of which consists of the input, output and computational tasks that process input and intermediate data. The component tasks of an application must be assigned to, and scheduled on, the processors in the system using techniques such as clustering (Ramamritham, 1995), list scheduling (Wang et al., 1992), or branch-and-bound strategies (Peng and Shin, 1989; Hou and Shin, 1997). Task assignments are governed by locality constraints that are

* This work was done while Jan Jonsson was at the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122, USA.

either strict (i.e., assignment of a task is known beforehand) or relaxed (i.e., there exist more than one assignment alternative for each task).

A real-time application is constrained to start and complete its execution within a given time span called the end-to-end deadline. The application has usually been decomposed into a set of sequential and/or parallel component tasks, often for reusability and/or parallelism/performance reasons. The E-T-E deadline must therefore be distributed over the component tasks; that is, a local arrival time and a deadline must be assigned to each component task for its scheduling.

Since the combined deadline-distribution and task-assignment problem is NP-hard, we present a heuristic approach for a distributed hard real-time system under relaxed locality constraints. Our approach is based on the slicing technique (Di Natale and Stankovic, 1994) in which the E-T-E deadline for each sequence of tasks in the application is decomposed into a set of non-overlapping task execution windows (each defined by an arrival time and a deadline). The distinct feature of the slicing technique is that it divides the overall problem into smaller problems which are solved locally and then combined to obtain a global solution, thereby drastically reducing the computational complexity.

In a distributed hard real-time system, task assignment and scheduling is usually assumed to be performed off-line in order to guarantee *a priori* the schedulability of each hard real-time task. The applications of interest in this paper are composed of sequential–parallel tasks with individual arrival times and deadlines, and precedence constraints specified. Our primary scheduling objective is to maximize the success ratio, or the ratio of the number of successfully-scheduled task sets to the total number of task sets considered.

The salient features of the proposed approach are demonstrated by using extensive simulations with randomly-generated task sets and a multiprocessor system of varying size. In particular, we evaluate the impact of variations in architecture and application properties on the success ratio for a baseline deadline-driven task assignment scheme and a non-preemptive, time-driven scheduling policy. The properties under investigation are system size, laxity of E-T-E deadline, and task execution-time distribution.[1] These properties are of significant interest because (1) schedulability may depend on how well the inherent application parallelism can be exploited on the available processors; (2) laxity directly controls the amount of extra time that will be available for distribution; and (3) under a non-preemptive scheduling policy, schedulability is directly affected by the distribution of task execution times.

The main contributions of this paper are as follows.

C1. The slicing technique is applied to distributed hard real-time systems with relaxed locality constraints. This is in sharp contrast to existing deadline-distribution techniques all of which are based on the knowledge of complete task assignment. Specifically, we evaluate communication-cost estimation techniques suitable for this type of systems, and find that the best approach for a distributed system is to always assume for deadline distribution that there is no communication cost in the system. This leaves the largest extra time to be distributed over component tasks.

C2. The proposed technique makes an order-of-magnitude improvement over, and perform much better than, the original slicing technique in (Di Natale and Stankovic, 1994) in terms of the success ratio. More specifically, we propose two adaptive metrics for determining each task deadline. The first, called ADAPT-G, has an $O(n^2)$ time-complexity for a system with n tasks, and accounts for average application parallelism and system size. The second metric, called ADAPT-L, has an $O(n^3)$ time-complexity and utilizes information on possible resource contention for individual tasks in the application. An extensive simulation study has shown the proposed metrics to outperform the non-adaptive counterparts over a wide range of system configurations. In particular, for small systems, ADAPT-L is found to outperform the non-adaptive metrics by as much as an order of magnitude. Furthermore, ADAPT-L makes a 300% improvement in performance over ADAPT-G.

The rest of the paper is organized as follows. Section 2 discusses the problem background and related work. Section 3 presents the system models used. Section 4 states the deadline-distribution problem and presents a basic algorithm to solve the problem. Section 5 describes our experimental setup and Section 6 deals with the experimental evaluation. Section 7 discusses complementary results and future work. Finally, Section 8 concludes the paper by summarizing the results obtained.

2. Background

2.1. Motivation

Many researchers have addressed the deadline-distribution problem (Di Natale and Stankovic, 1994; Garey et al., 1981; Chetto et al., 1990; Kao and Garcia-Molina, 1993; 1994; Abdelzaher and Shin, 1995; Gutiérrez García and González Harbour, 1995; Bettati and Liu, 1992; Saksena and Hong, 1996a,b), all under a common assumption that task assignments are made and known *a priori* (i.e., strict locality constraints). In many real-time systems, however, only a small percentage of tasks are governed by strict locality constraints, typically imposed by their resource needs in physical proximity, like sensors and actuators. The constraints on the remaining task assignments are not strict, meaning that information on task execution times and intertask communication cost is not always known *a priori*. For a homogeneous system and negligible intertask communication cost, this poses little difficulty as all processors are interchangeable. For a heterogeneous system, however, tasks may have different execution times on different processors, and the deadline-distribution problem under relaxed locality constraints is much harder to solve. Furthermore, task assignment requires knowledge of deadlines of individual tasks. Deadline distribution using conventional techniques, on the other hand, can be possible only if task assignments are completely known. Thus, there exists a circular dependency between deadline distribution and task assignment, making the combined deadline-distribution and task-assignment problem even harder to solve. The task-assignment problem is, in general, NP-complete (Garey and Johnson, 1979), and good solutions to this problem must, therefore, be found via sub-optimal heuristic algorithms.

Our solution is based on the slicing technique (Di Natale and Stankovic, 1994), the best existing deadline-distribution method for distributed real-time systems. The distinguishing feature of this technique is that the E-T-E deadline for each application is decomposed into a set of non-overlapping task execution windows called slices. Since slices are non-overlapping, each task in the application will complete before its successor task is released, thus providing the following important properties.

P1. Sequential tasks executing on two different processors can be scheduled independently of each other. This is useful in heterogeneous systems where different processors use different scheduling policies. It also allows for parallel scheduling of sequential tasks, enhancing scheduling performance in those systems equipped with on-line scheduling capabilities (Cheng et al., 1986).

P2. One can eliminate the release-time jitter of each task caused by precedence constraints. As discussed in, for example Audsley et al. (1993) and Di Natale and Stankovic (1995), uncontrolled release jitters can negatively affect the schedulability of real-time applications. In addition, the stability of a computer-controlled system may be affected by the release jitter.

2.2. Related Work

The slicing technique proposed in Di Natale and Stankovic (1994) assigns slices to tasks using the critical-path concept. The strategy used for finding slices is to determine a critical path in the task graph that minimizes the overall laxity of the path. Two basic metrics, previously proposed in Kao and Garcia-Molina (1993, 1994), were used for evaluating paths in the task graph: one assigns a task deadline based on its execution time, and the other is based on the number of tasks in the critical path.

The slicing technique is optimal in the sense that it maximizes the minimum task laxity in an application. However, the optimality applies only if task assignments and communication costs are completely known *a priori*.

A traditionally used technique for deadline assignment is the one proposed by Garey et al. (1981), and later by Chetto et al. (1990). This technique assigns the largest possible execution window to each task while taking into account any precedence constraints and E-T-E deadlines. Because this technique was originally designed for single-processor systems, it lacks the important properties, P1 and P2, of the slicing technique that are necessary for distributed real-time systems.

Kao and Garcia-Molina presented multiple strategies for distributing E-T-E deadlines over sequential (Kao and Garcia-Molina, 1993) and sequential–parallel (Kao and Garcia-Molina, 1994) tasks. They classify deadline-distribution strategies as being local or global. The local strategies derive task deadlines from the E-T-E deadline based on task set characteristics (similar to the slicing technique), while the global ones use simpler functions of the global E-T-E deadline (such as the one in Garey et al. (1981) and Chetto et al. (1990)). Both types of strategies are aimed at systems with complete *a priori* knowledge of task–processor assignment.

Abdelzaher and Shin (1995) proposed a technique for integrated scheduling of tasks and messages in a distributed real-time system. The technique is based on a branch-and-bound algorithm that improves upon a given task assignment with local deadlines already assigned, for deadline-based task and message priorities. Task execution windows have adjustable positions in time, rather than static as in the slicing technique, which caters for better exploitation of processors.

In Gutiérrez García and González Harbour (1995), Gutiérrez García and González Harbour proposed an approach that derives deadlines for preemptive, deadline-monotonic task scheduling. The strategy is based on a heuristic iterative approach that, given an initial local deadline assignment (from which initial task priorities are derived), finds an improved deadline assignment in reasonable time. For each iteration a new deadline assignment is calculated based on a metric that measures by ''how much'' schedulability has failed. This approach assumes that the application consists of purely sequential tasks and that task assignment is known beforehand.

Bettati and Liu (1992) presented an approach for scheduling of pre-allocated flow-shop (sequential) tasks. Similar to a variant of the slicing technique, local deadlines are assigned by distributing E-T-E deadlines evenly over tasks. After deadlines have been assigned, tasks are scheduled non-preemptively using a deadline-based priority scheme. In this method, the simplifying assumption is made that execution times are either identical for all tasks or identical for all tasks assigned to the same processor.

Saksena and Hong (1996a,b) proposed a deadline-distribution approach for pre-allocated tasks. The E-T-E deadline is expressed as a set of local deadline-assignment constraints. Given a set of local deadline assignments, the technique calculates the largest value of a scaling factor (which is applied to the task execution times) that still makes the tasks schedulable under a preemptive task scheduling policy. The local deadline assignment is then chosen to maximize the largest value of the scaling factor.

A flexible approach has been proposed by Tindell et al. (1992). Based on the simulated annealing principle, the approach uses a combined assignment and scheduling algorithm for systems with both relaxed and strict locality constraints. No explicit attempt is made in this approach to solve the deadline-distribution problem; instead, it is assumed that the deadline of a task is equal to its period (minus possible overhead to account for any interprocessor communication overhead). The approach in Tindell et al. (1992) has been successfully demonstrated for preemptive, deadline-monotonic task scheduling on a distributed architecture.

The approach most comparable to ours is the simulated-annealing approach (Tindell et al., 1992), in the sense that it, too, can handle any combination of relaxed and strict locality constraints. This is in sharp contrast to the other approaches described above where task assignment is assumed to be known in advance. However, the work in Tindell et al. (1992) does not address the problem of deriving local task deadlines in a precedence-constrained application, which is the focus of our work. On the other hand, the simulated-annealing approach can be configured to optimize many different objectives, not only schedulability or maximum task lateness, and can also be used to find optimal solutions. Such flexibility is not sought in our approach, but it is instead powerful in the sense that application parallelism can be accounted for during deadline distribution so as to increase the likelihood of successfully scheduling the application. This is not the case for the original slicing technique (Di Natale and Stankovic, 1994) or for the approaches in Bettati and Liu, (1992) and Kao and Garcia-Molina (1993). Moreover, our approach is not designed with a specific task scheduling policy in mind, as is the case for the approaches in (Abdelzaher and Shin, 1995; Gutiérrez García and González Harbour, 1995; Saksena and Hong, 1996b). This, naturally, makes our approach more widely applicable.

3. System Model

3.1. Architecture Model

We consider a multiprocessor architecture with a set $\mathbf{P} = \{p_q : 1 \leq q \leq m\}$ of schedulable computing resources (processors) and an interconnection network with a number of communication resources (links). The processors are homogeneous in the sense that they have identical hardware configurations in terms of processing speed, instruction pipeline, and cache/primary memory resources. This means that each task takes the same amount of time to execute on any processor.

The processors are interconnected via an arbitrary topology formed with dedicated as well as shared links. Two tasks residing on the same processor communicate by accessing shared memory and the communication cost is assumed to be negligible. The communication cost between two tasks on different processors is expressed as the product of the length of a message and a nominal communication delay. The nominal delay is an upper-bounded and predictable worst-case communication delay that reflects the scheduling strategy of the underlying interconnection network. Communication in the network is assumed to take place concurrently (e.g., using a communication co-processor (Rexford et al., 1996)) with task execution on the processors.

We will assume that the system maintains a global system time that is discrete (Kopetz, 1992) and measured in time units indexed by the natural numbers, $t \in \mathbf{N}$. Without loss of generality, we assume that task execution begins and ends at time unit boundaries, and that application timing parameters are expressed as an integer multiple of time units. A consistent view of system time is maintained in each processor by means of a system-wide clock synchronization mechanism.

3.2. Task Model

We consider a real-time application that consists of a set $\mathbf{T} = \{\tau_i : 1 \leq i \leq n\}$ of tasks. Each task $\tau_i \in \mathbf{T}$ is characterized by a 4-tuple $\langle c_i, \phi_i, d_i, T_i \rangle$, which we will refer to as the static task parameters. The worst-case execution time c_i is an upper bound on the execution time of the task and includes various architecture overhead such as the cost for cache memory misses, pipeline hazards, and context switches. It also includes the cost for packetizing and depacketizing messages in case of communicating tasks. The phasing ϕ_i is the earliest time at which the first invocation of the task will occur, measured relative to some fixed origin of time. The relative deadline d_i is the amount of time within which the task must complete its execution.

The task period T_i is the time interval between two consecutive invocations of the task. Let τ_i^k denote the k-th invocation of τ_i, $k \in \mathbf{Z}^+$. The dynamic behavior of τ_i^k is characterized by the dynamic task parameters (a_i^k, D_i^k), where the absolute arrival time $a_i^k = \phi_i + T_i(k-1)$ is the earliest time at which τ_i^k is allowed to start its execution, and the absolute deadline $D_i^k = a_i^k + d_i$ is the latest time by which τ_i^k must complete its execution. Whenever it is clear from the context what invocation is being referred to, we will drop the invocation superscripts and write (a_i, D_i) for the dynamic parameters of τ_i.

Precedence constraints between tasks in a task set (typically to model independent applications) are represented by an irreflexive partial order $\prec$ over $\mathbf{T}$. If task τ_j cannot begin its execution until task τ_i has completed its execution, we write $\tau_i \prec \tau_j$. In this case τ_i is said to be a predecessor of τ_j, and, conversely, τ_j a successor of τ_i. In addition, whenever $\tau_i \prec \tau_j$ and the condition $\neg(\exists \tau_k : (\tau_i \prec \tau_k) \wedge (\tau_k \prec \tau_j))$ holds, we write $\tau_i \prec\!\cdot\, \tau_j$. In this case, τ_i is said to be an immediate predecessor of task τ_j and τ_j an immediate successor of τ_i. A task which has no predecessors is called an input task, and a task which has no successors is called an output task.

A task chain is defined as a task followed by a series of immediate successors. The length of a task chain is the sum of the worst-case execution times of all tasks in the chain. The static level $SL(\tau_i)$ for a task τ_i is the length of the longest task chain that starts with τ_i and ends with an output task. A path Φ is a task chain that begins with an input task and ends with an output task.

Data communication in the application (and between applications) can be embedded in the precedence constraints between tasks, and can be implemented with general communication primitives such as SEND–RECEIVE–REPLY and QUERY–RESPONSE (Peng and Shin, 1989). Let $m_{i,j}$ be the size of a message, or the amount of data, transferred between task τ_i and task τ_j. The worst-case communication cost depends on various factors, such as task assignment, message size, communication medium bandwidth, message dispatching strategy, and nominal (contention-free) communication link delay.

The computational and communication demands of all tasks in $\mathbf{T}$, and the precedence constraints among them, are represented by a directed acyclic graph $\mathbf{G} = (\mathbf{N}, \mathbf{A})$ called a task graph. $\mathbf{N}$ is a set of nodes representing the tasks in $\mathbf{T}$. $\mathbf{A}$ is a set of directed arcs representing the precedence constraints between the tasks in $\mathbf{T}$: if $\tau_i \prec\!\cdot\, \tau_j$ then $(\tau_i, \tau_j) \in \mathbf{A}$. Each node in $\mathbf{N}$ is annotated with a non-negative weight representing the execution time of the corresponding task. For those arcs in $\mathbf{A}$ that represent communication channels, a non-negative weight is used for representing the message size.

3.3. Multiprocessor Scheduling

A non-preemptive,[2] time-driven multiprocessor schedule for a task set $\mathbf{T}$ and a multiprocessor architecture $\mathbf{P}$ is the mapping of each task $\tau_i \in \mathbf{T}$ to a start time s_i and a processor $p(\tau_i) \in \mathbf{P}$. The task is then scheduled to run without preemption on processor

$p(\tau_i)$ in the time interval $[s_i, f_i]$, with its finish time being $f_i = s_i + c_i$. The time interval $[a_i, D_i]$, denoted by w_i, is called the execution window of τ_i. For periodic tasks, the static task parameters are assumed to satisfy $d_i \leq T_i$, that is, the execution windows of two consecutive invocations of the same task cannot overlap in time. Furthermore, the execution time c_i cannot exceed the size $|w_i|$ of any execution window. Note that we assume the use of a static assignment, that is, all invocations of τ_i are executed on the same processor $p(\tau_i)$.

The schedule is said to be valid if (i) the conditions $s_i \geq a_i$ and $f_i \leq D_i$ are satisfied for each task τ_i, and (ii) all precedence constraints defined by the partial order $\prec$ over $\mathbf{T}$ are observed. A task set is said to be feasible if there exists a valid schedule for the task set. We say that a task set is schedulable by a scheduling algorithm if it produces a valid schedule for the task set.

A task τ_i is said to be ready at time t if its immediate predecessors have finished their execution. A task τ_i is said to be released at time t if (i) it is ready, (ii) $a_i \leq t$, and (iii) the task has received all necessary data from its immediate predecessors. A processor p_q is said to be available at time t if it has executed all tasks assigned to, and scheduled on, that processor.

In order to handle a periodic task system, we only need to analyze the task behavior within a specific period, called the planning cycle, P, that repeats itself throughout the life-time of the system. For a set of tasks with an identical arrival time, the planning cycle can be found as follows. Without loss of generality, assume that $\forall \tau_i : a_i = 0$. We then choose $P = [0, L)$, where L, the length of the planning cycle, is defined as the least common multiple (LCM) of $\{T_i : \tau_i \in \mathbf{T}\}$. Within P, τ_i will be invoked L/T_i times. For a set of tasks with arbitrary arrival times, we find the planning cycle as follows (Leung and Merrill, 1980). Without loss of generality, one can assume that $\min\{a_i : \tau_i \in \mathbf{T}\} = 0$. Furthermore, define $a = \max\{a_i : \tau_i \in \mathbf{T}\}$. We then choose $P = [0, a + 2L)$.

Note that a real-time application may consist of applications of different periods[3] that exchange messages using the communication primitives mentioned in Section 3.2. Thus, what is actually being scheduled are one or more invocations of each task within a planning cycle.[4]

4. Deadline Distribution

4.1. Problem Statement

Given an E-T-E deadline D_α and a corresponding input–output task pair $(\tau_{\alpha_1}, \tau_{\alpha_2})$, we want to partition (distribute) D_α into an arrival time a_i and a relative deadline d_i for task τ_i in the task graph in such a way that the path constraint

$$\sum_{\tau_i \in \Phi} d_i \leq D_\alpha \tag{1}$$

is satisfied for every path Φ between τ_{α_1} and τ_{α_2}.

For a given path Φ and its associated E-T-E deadline D_Φ, the surplus for the deadlines of tasks in Φ will be taken from the overall laxity X_Φ of Φ, defined as

$$X_\Phi = D_\Phi - \sum_{\tau_i \in \Phi} c_i \tag{2}$$

4.2. *Quality Assessment*

A solution to the deadline-distribution problem cannot be accepted simply because it satisfies the path constraint defined above. One also has to consider the practical issue of schedulability: the relative deadline of a task must be derived in such a way that the task is likely to be feasibly scheduled. If one deadline-distribution strategy can feasibly schedule more task sets than another strategy, the first strategy is superior to the second one. Therefore, our primary performance measure for a deadline distribution is the success ratio. If a deadline-distribution strategy can aid in finding feasible schedules for x of the considered y task sets, its success ratio is (x/y).

When E-T-E deadlines are loose enough to guarantee a near 100% success ratio, a secondary performance measure should be used to assess the quality of different deadline-distribution strategies. Two important measures that are often used for this purpose are the minimum laxity and the maximum lateness taken over all tasks in the system. The laxity $X_i = d_i - c_i$ is the maximum amount of time that the execution of task τ_i can be delayed in its execution window without missing its absolute deadline. The laxity is determined before the task is scheduled and, thus, suggests how much contention for the processor the task can withstand. The lateness $L_i = f_i - D_i$ is the amount by which task τ_i misses its deadline, that is, a non-positive quantity for a valid schedule. The lateness is determined after scheduling the tasks, and, thus, indicates the quality of the schedule. The maximum lateness refers to the lateness of only one task (the one with its lateness closest to 0), and, thus, indicates ''how far'' the schedule is from infeasibility and how much additional background workload the schedule can handle.

4.3. *The Slicing Technique*

We solve the deadline-distribution problem by significantly improving the slicing technique in Di Natale and Stankovic (1994). The fundamental concept in the slicing technique is a critical path in a task graph that attains the minimum/maximum value of a performance measure under consideration. Correct identification of a critical path is crucial for the quality of deadline distribution and system schedulability. In order to assess the criticality of each path in the task graph, a critical-path metric is used.

When a critical path has been identified, the E-T-E deadline is distributed over the tasks in the critical path. This is done by assigning slices, non-overlapping execution windows, of the E-T-E deadline to each of the tasks in the critical path. The slices are derived subject to the constraint that the arrival time of a task must be equal to the absolute deadline of its immediate predecessor in the critical path. While the execution windows of tasks in the same path cannot overlap, the execution windows of tasks in different paths may overlap and thus are subject to contention for available processors. Therefore, the size of each task slice must be derived in such a way that the task can be scheduled to meet its timing constraint even in the presence of other, overlapping, task slices. This can be achieved by assigning an ample laxity to each task.

For a system with complete *a priori* information on task–to–processor assignment and interprocessor-communication cost, and moderate contention over resources, the best critical path can easily be found as described in Di Natale and Stankovic (1994). However, when the assignment is not entirely fixed or known—as is often the case—finding the best critical path is no longer an easy task because it is not yet known what pairs of tasks will suffer interprocessor-communication overhead or experience resource contention. The deadline-distribution algorithm must, therefore, rely on the prediction of the possibly-best critical path.

Our technique for making a good prediction of the critical path is based on the following two observations. First, in systems where the degree of application parallelism exceeds the number of processors available, we may assign longer slices to tasks with longer execution times. The rationale behind this is that tasks with longer execution times are likely to be most vulnerable in the case of heavy resource contention. Second, it is a good strategy to assume—during deadline distribution—that there will be no communication cost between tasks whose processor assignments are not known. The intuition behind this is that many task assignment and scheduling algorithms tend to cluster tasks that communicate heavily (Ramamritham, 1995). Furthermore, in many distributed real-time applications (such as control systems), the intertask communication volume is quite low (Raji, 1994), so the interprocessor communication cost is low relative to task execution times. In Section 6.4, we will show that, even in the case of significant communication cost, it is better to assume no communication cost, as this will yield the largest amount of overall laxity to distribute over the application tasks.[5]

4.4. The Deadline-Distribution Algorithm

Figure 1 shows a pseudo-code form of the algorithm for distributing E-T-E deadlines over the component tasks according to the slicing technique. The algorithm takes a task graph as the input and produces an annotated task graph containing the information about task arrival times and relative deadlines. The various steps of the algorithm are described below in detail. In the algorithm, we assume that task slices are determined using a critical-path metric R.

Initialize task set (Step 1): We assume that all input and output tasks have already been assigned appropriate arrival times and E-T-E deadlines, respectively, according to the temporal requirements of the application. All tasks in the task graph are then inserted into a task set Π that represents all tasks not yet assigned arrival times and deadlines. The time-complexity of this step is $O(|\mathbf{N}|)$. This does not include the time for initializing data structures used by the critical-path metric R.

Find a critical path (Step 3): The breadth-first traversal of the task graph determines a critical path Π among all potential paths for the tasks in Φ. The critical path is identified

```
Algorithm SLICING:

1.  initialize set $\Pi$ with all tasks in the task graph;
2.  while $\{\Pi \neq \emptyset\}$ loop
3.    find a critical path $\Phi$ in $\Pi$ that minimizes metric $R$;
4.    distribute the E-T-E deadline of $\Phi$ by assigning
        arrival times and absolute deadlines to the tasks in $\Phi$;
5.    for $\{$each task $\tau$ in $\Phi\}$ loop
6.      for $\{$each immediate predecessor $\tau_p$ of $\tau\}$ loop
7.        assign an absolute deadline to $\tau_p$ that is equal to the arrival time of $\tau$;
8.      end loop;
9.      for $\{$each immediate successor $\tau_s$ of $\tau\}$ loop
10.       assign an arrival time to $\tau_s$ that is equal to the absolute deadline of $\tau$;
11.     end loop;
12.   end loop;
13.   remove all tasks in $\Phi$ from $\Pi$;
14. end loop;
```

Figure 1. The deadline-distribution algorithm.

using the metric R. Ties among those paths with identical metric values are broken arbitrarily. Critical-path metrics R suitable for our purpose will be discussed in Section 4.5. The time-complexity of a breadth-first traversal of the task graph is $O(|\mathbf{N}| + |\mathbf{A}|)$ (Cormen et al., 1990). Therefore, for any acyclic task graph, the total time-complexity of this step, taken over all iterations of the main loop, is $O(|\mathbf{N}|^2)$. This does not include the time for evaluating the critical-path metric R.

Distribute the E-T-E deadline (Step 4): The E-T-E deadline D_Φ of the critical path Φ found in Step 3 is distributed over the tasks in Φ. The deadline distribution is subject to the constraint that the arrival time of a task in Φ must be equal to the absolute deadline of its immediate predecessor in Φ. Thus, all tasks in the path will be assigned slices of the E-T-E deadline. The total time-complexity of this step is $O(|\mathbf{N}|)$, since a task will only be part of a critical path once (recall that the task graph $\mathbf{G}$ is an acyclic graph and, hence, no loops exist in the graph).

Attach the remaining tasks (Steps 5–12): The tasks in Φ now constitute a ''spine'' to which the remaining tasks must attach, that is, adapt their arrival times and absolute deadlines. Therefore, the arrival time of each task not in Φ is set to the latest absolute deadline of any immediate predecessor task in Φ. Similarly, the absolute deadline of each task not in Φ is set to the earliest arrival time of any immediate successor task in Φ. The total time-complexity of this step is $O(|\mathbf{N}| + |\mathbf{A}|)$, since a task will be part of a critical path only once and the arcs connecting the task with its immediate predecessors and successors will be traversed only once.

Remove critical-path tasks (Step 13): The tasks in Φ are removed from Π to mark that they have been assigned arrival times and deadlines. The arrival times and deadlines of those tasks not in Φ now constitute new E-T-E deadlines in Π. The total time-complexity of this step is $O(|\mathbf{N}|)$.

Repeat until tasks are exhausted (Steps 2 and 14): The main loop in the algorithm is repeated until no tasks are left in Π.

Adding up the time-complexities of the steps described above leads to the worst-case time-complexity of $O(n^2)$, not counting the computational overhead associated with the critical-path metric R. The impact of R on the practicality of the slicing technique will be discussed in Section 7.2.

4.5. Critical-Path Metrics

The authors (Di Natale and Stankovic, 1994) used two metrics with the slicing technique. The first metric, the normalized laxity ratio (NORM), is the ratio of the overall laxity to the sum of the execution times of all tasks in a path Φ. With this metric, laxity is assigned in proportion to the task execution time. The metric value R_{NORM} and the relative deadline d_i for each task τ_i are consequently defined as

$$R_{\text{NORM}} = \left(D_\Phi - \sum_{\tau_i \in \Phi} c_i \right) \Big/ \sum_{\tau_i \in \Phi} c_i \tag{3}$$

$$d_i = c_i (1 + R_{\text{NORM}}) \tag{4}$$

The second metric in Di Natale and Stankovic (1994), the pure laxity ratio (PURE), is the ratio of the overall laxity to the number of tasks n_Φ in a path Φ. With this metric, all tasks in Φ are assigned an equal share of laxity. The metric value R_{PURE} and the relative deadline d_i for task τ_i are thus

$$R_{\text{PURE}} = \left(D_\Phi - \sum_{\tau_i \in \Phi} c_i \right) \Big/ n_\Phi \tag{5}$$

$$d_i = c_i + R_{\text{PURE}} \tag{6}$$

In the simulations that follow, we will show that neither the NORM or PURE metric is suitable for systems with relaxed locality constraints due to their lack of information on contention over resources. To remedy this problem, we introduce two concepts to improve the performance of the slicing technique: execution-time threshold and virtual execution time. The execution-time threshold is a mechanism for guaranteeing only certain tasks to be allotted extra laxities. By using the execution-time threshold to filter out those tasks with large enough execution times, we can improve the performance when task graph parallelism cannot be fully exploited. The purpose of a virtual execution time is to make a task appear computationally more expensive than it actually is. By using virtual execution times that are larger than the real execution times, the deadline-distribution algorithm will allocate more laxity to those tasks for which the real execution time is equal to, or larger than, the given execution time threshold.

Based on these two concepts, we now introduce the threshold laxity ratio (THRES) metric, which is similar to the PURE metric but with a virtual execution time $\hat{c}$, as opposed to the real execution time c. The virtual execution time for task τ_i is defined as

$$\hat{c}_i = \begin{cases} c_i & \text{if } c_i < c_{\text{thres}} \\ c_i(1 + k_{\text{S}}) & \text{if } c_i \geq c_{\text{thres}} \end{cases} \tag{7}$$

where c_{thres} is the execution time threshold and k_{S} is a surplus factor representing the amount of laxity by which the real execution time should be increased for those tasks whose execution times exceed c_{thres}.

As our simulation results will show later, it is very difficult to attain a consistently high performance for THRES with a fixed value of k_S. We therefore propose two improvements of the threshold-based metric where the amount of assigned laxity is not fixed, but will adapt itself to the degree of task graph parallelism that can be exploited.

The first of these adaptive metrics, which we call the globally-adaptive laxity ratio (ADAPT-G), is similar to the THRES metric but with a different definition of the virtual execution time:

$$\hat{c}_i = \begin{cases} c_i & \text{if } c_i < c_{\text{thres}} \\ c_i(1 + k_G \xi / m) & \text{if } c_i \geq c_{\text{thres}} \end{cases} \tag{8}$$

where k_{G} is the global adaptivity factor, ξ is the average task graph parallelism, and m is the number of processors in the system. The average task graph parallelism is defined as the application workload divided by the length of the longest path in $\mathbf{T}$:

$$\xi = \frac{\sum_{\tau_i \in \mathbf{T}} c_i}{\max\limits_{\tau_i \in \mathbf{T}} \{SL(\tau_i)\}} \tag{9}$$

The second adaptive metric is the locally-adaptive laxity ratio (ADAPT-L) metric, which is an improvement of ADAPT-G in the sense that ADAPT-L is able to identify the available parallelism that affects only a certain task. For ADAPT-L, the virtual execution time is defined as:

$$\hat{c}_i = \begin{cases} c_i & \text{if } c_i < c_{\text{thres}} \\ c_i(1 + k_{\text{L}} |\Psi_i| / m) & \text{if } c_i \geq c_{\text{thres}} \end{cases} \tag{10}$$

where k_L is the local adaptivity factor, Ψ_i is the parallel set of τ_i, and m is the number of processors in the system. The parallel set Ψ_i is the set of tasks that are potential candidates for executing in parallel with τ_i, that is, those tasks that are neither predecessors nor successors of τ_i.

ADAPT-L introduces the extra complexity to the deadline-distribution algorithm that a parallel set must be calculated for each task. The parallel set Ψ_i for task τ_i can easily be found by first constructing the transitive closure $\mathbf{G}^*$ for the task graph $\mathbf{G}$, and then in Ψ_i include only those tasks that are neither reachable from τ_i, nor can reach τ_i. The transitive closure $\mathbf{G}^*$ can be constructed during the initialization phase (Step 1) of the deadline-distribution algorithm and takes time $O(n^3)$ (Cormen et al., 1990).

4.6. An Application Case Study

Figure 2 illustrates the task graph of a small control application running on a single processor. The application consists of five tasks: a Sensor Task τ_1, an Actuator Task τ_5, and three Data Manipulation Tasks τ_2, τ_3, and τ_4. In the graph, each node is labeled with the corresponding task identifier τ_i and execution time c_i, and each arc represents a message transmission between two tasks. Since all communication is performed on a single-processor system, any communication cost can be ignored and, hence, the arcs are treated as pure precedence constraints. For the Actuator Task, the absolute deadline, $D_5 = 21$, is given in conjunction with the corresponding node. For the Sensor Task, an arrival time, $a_1 = 0$, is given. Clearly, this application will experience significant resource contention during scheduling, owing to the fact that tasks τ_2, τ_3, and τ_4 can potentially execute in parallel while only one processing resource is available. The parallel sets, Ψ_i, of these tasks all have a size $|\Psi_i| = 2$. The parallel sets of the other tasks, τ_1 and τ_5, are empty and thus have a size $|\Psi_i| = 0$. The mean task execution time of the application is $c_{\text{mean}} = 3.4$.

To demonstrate how the choice of deadline assignment is crucial for the example application's schedulability, we have applied the slicing technique using the PURE and ADAPT-L metrics, representing the non-adaptive and adaptive metrics, respectively. For ADAPT-L, we have used the parameter values $c_{\text{thres}} = 1.0 * c_{\text{mean}}$ and $k_L = 1.0$. With this execution-time threshold, only tasks τ_3 and τ_4 will have virtual execution times (i.e. $\hat{c}_i = 12$) that are different from the real execution times.

The results from applying the slicing technique to the example application are shown in Table 1. The algorithm in Figure 1 takes three loop iterations to complete a deadline assignment for the example application. In each iteration of the algorithm, the path with the smallest value of R is chosen (indicated in bold). The execution windows, $w_i = (a_i, D_i)$, resulting from the deadline assignment are also shown in the table. Here, we can identify the importance of the choice of critical-path metric, R. With PURE, the accumulated execution time $(c_2 + c_3 + c_4 = 11)$ of the parallel tasks is much larger than

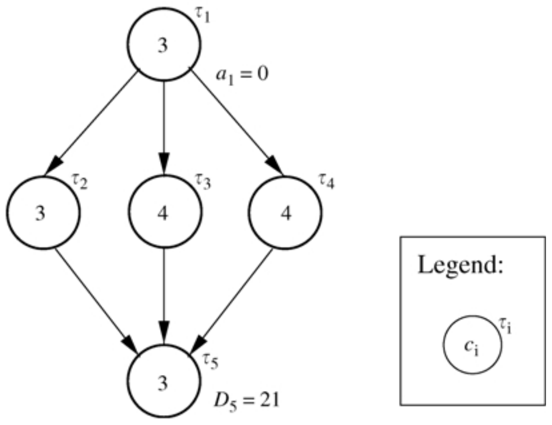

Figure 2. A simple control application.

Table 1. Deadline assignment for the simple control application.

				(a_i, D_i)				
Metric	Loop #	Path	R	τ_1	τ_2	τ_3	τ_4	τ_5
PURE	1	$\tau_1 \rightarrow \tau_2 \rightarrow \tau_5$	4.00					
		$\tau_1 \rightarrow \tau_3 \rightarrow \tau_5$	**3.67**	(0.00, 6.67)	—	(6.67, 14.33)	—	(14.33,21.00)
		$\tau_1 \rightarrow \tau_4 \rightarrow \tau_5$	3.67					
	2	τ_2	4.67					
		τ_4	**3.67**	(0.00, 6.67)	—	(6.67, 14.33)	(6.67, 14.33)	(14.33, 21.00)
	3	τ_2	**4.67**	(0.00, 6.67)	(6.67, 14.33)	(6.67, 14.33)	(6.67, 14.33)	(14.33, 21.00)
ADAPT-L	1	$\tau_1 \rightarrow \tau_2 \rightarrow \tau_5$	4.00					
		$\tau_1 \rightarrow \tau_3 \rightarrow \tau_5$	**1.00**	(0.00, 4.00)	—	(4.00, 17.00)	—	(17.00, 21.00)
		$\tau_1 \rightarrow \tau_4 \rightarrow \tau_5$	1.00					
	2	τ_2	10.00					
		τ_4	**1.00**	(0.00, 4.00)	—	(4.00, 17.00)	(4.00, 17.00)	(17.00, 21.00)
	3	τ_2	**10.00**	(0.00, 4.00)	(4.00, 17.00)	(4.00, 17.00)	(4.00, 17.00)	(17.00, 21.00)

the size $(|w_i| = |(6.67, 14.33)| = 7.67)$ of their common execution window. This means that these tasks cannot all be scheduled to meet their deadlines on the single processor. For ADAPT-L, on the other hand, the size $(|w_i| = |(4.00, 17.00)| = 13)$ of the common execution window is large enough to allow for meeting all parallel tasks' deadlines—even for the case of a single processor.

Although the results from our case study suggest that the proposed adaptive metrics can improve scheduling performance, it is still necessary to study their impact on a larger application domain. Therefore, in the following sections, we show that both ADAPT-G and ADAPT-L outperform the non-adaptive metrics with respect to the success ratio as well as maximum task lateness. Furthermore, we show that the adaptive metrics also exhibit a very robust behavior over a variety of parameter settings.

5. Experimental Setup

5.1. *System Architecture*

We have used an experimental platform based on a homogeneous multiprocessor architecture with a shared bus. The system size ranges from two to eight processors. The shared bus is time-multiplexed in such a way that the communication cost between two processors is one time unit per transmitted data item.[6]

5.2. *Workload*

In all experiments, a set of 1024 task graphs were generated using a random task graph generator. Each task graph contained between 40 and 60 tasks without any imposed task-assignment restrictions (i.e. relaxed locality constraints). Task execution times were chosen randomly assuming a uniform distribution with a mean execution time, c_{mean}, of 20 time units. To mimic different application scenarios, task execution times were chosen based on the execution-time distribution (ETD), the maximum percentage deviation of a task's execution time from the mean execution time, c_{mean}. Thus, for a given ETD, the task execution times were chosen at random to be in the range $[c_{\text{mean}}(1 - \text{ETD}), c_{\text{mean}}(1 + \text{ETD})]$. To evaluate the performance of the metrics under variable tight timing constraints, an E-T-E deadline was chosen for each input–output task pair based on the overall laxity ratio (OLR), the ratio of the E-T-E deadline to the accumulated task graph workload. The precedence constraints in the task graph were also randomly generated. The number of immediate successors/predecessors of each task was chosen at random to be in the range of one to three, and the depth of the task graph was chosen at random to be between eight and 12 levels. The number of data items in each message passed between a pair of tasks was chosen in such a way that the communication-to-computation cost ratio (CCR) of the average message communication cost to the average task execution time corresponded to 0.1.

5.3. *Task Assignment and Scheduling*

We have used a baseline task assignment and scheduling strategy based on a list scheduling version of the earliest-deadline-first (EDF) algorithm. For each scheduling step, our EDF algorithm selected one task, among all ready tasks, that had the closest absolute deadline. Then the selected task was scheduled on the available processor that yielded the earliest start time, taking into account possible communication cost[7] and the arrival time constraints of the task. The set of ready tasks was updated with the immediate successors of the scheduled task and the algorithm repeated until either all tasks were successfully scheduled or a scheduling attempt failed. The complexity of this algorithm is $O(n^2 * m)$ for a system with n tasks and m processors.

5.4. *Experimental Platform*

All modeling and simulations in the experiments were performed within GAST (Jonsson, 1997, 1998), a flexible tool for evaluation of uniprocessor and multiprocessor scheduling techniques. This environment allows us to perform experiments with parametrized models of the application and the architecture on an arbitrary granularity.

6. Experimental Evaluation

We evaluated the performance of the metrics presented in Section 4.5 using the deadline-distribution algorithm in Figure 1 and the experimental setup in Section 5. Unless otherwise noted, all simulations were performed with $\text{ETD} = 25\%$ and $\text{OLR} = 0.8$. The following default parameter values were used for the adaptive metrics during the experiments: $c_{\text{thres}} = 1.0 * c_{\text{mean}}$, $k_S = 1.0$, $k_G = 1.5$, $k_L = 0.2$. In Section 6.6, these values are shown to yield the most robust performance for a wide range of applications and architectures.

In all the plots presented, every reported value is an average of the observed performance measure, taken over the set of simulation runs for each parameter combination. With the chosen number of simulation runs, a 99% confidence level was obtained for a maximum error within 0.5% of the average values reported.

6.1. Effect of System Size on the Performance

Since one of the goals is to find metrics that perform well in the presence of heavy processor contention, we would like to study how the slicing metrics will behave for different system sizes. By varying the number of processors, we can identify the capability of each metric in exploiting the available application parallelism on the platform architecture. Intuitively, all metrics should perform worse for small system sizes than for a larger system. For a small system where the application parallelism exceeds the number of available processors, the contention between tasks over a few available processors forces multiple tasks to be scheduled within overlapping execution windows on the same processor. As the system size increases more parallelism in the task graph can be exploited, decreasing the contention for processors. Consequently, the success ratio will increase with increasing system size, until a point at which all generated task graphs are successfully scheduled. This behavior is corroborated by the results plotted in Figure 3 which show the success ratio as a function of system size assuming a fixed OLR.

Under a deadline-driven scheduling strategy, tasks with longer execution times will be the most susceptible to processor contention as shorter tasks are more likely to have shorter deadlines and thus will be scheduled earlier. So, longer tasks should be assigned more laxity to compensate for this imbalance. Figure 3 shows how the PURE metric, with its equal-share distribution strategy, exhibits the worst performance among all metrics. For example, only 35% of all scheduling attempts succeed on a system with four processors when PURE is used. The performance of the NORM metric is significantly better, due mainly to its strategy to assign deadlines in proportion to task execution times. This approach gives tasks with longer execution times more laxity and hence a better chance to be scheduled feasibly. For the four-processor system, more than 65% of all scheduling attempts succeed when NORM is used.

The threshold-based strategy of THRES is an alternative approach of assigning longer deadlines to tasks with longer execution times. THRES exhibits comparable performance to NORM for the assumed values of OLR and ETD. However, as we will demonstrate later, the performance of THRES is not very robust under varying application properties. For example, it performs better than NORM for smaller values of OLR but worse for larger values of ETD.

The figure clearly illustrates that the non-adaptive metrics (PURE, NORM, and THRES) do not perform well for small systems. For example, the fraction of successful scheduling attempts does not exceed 30% for a system with three processors for any non-

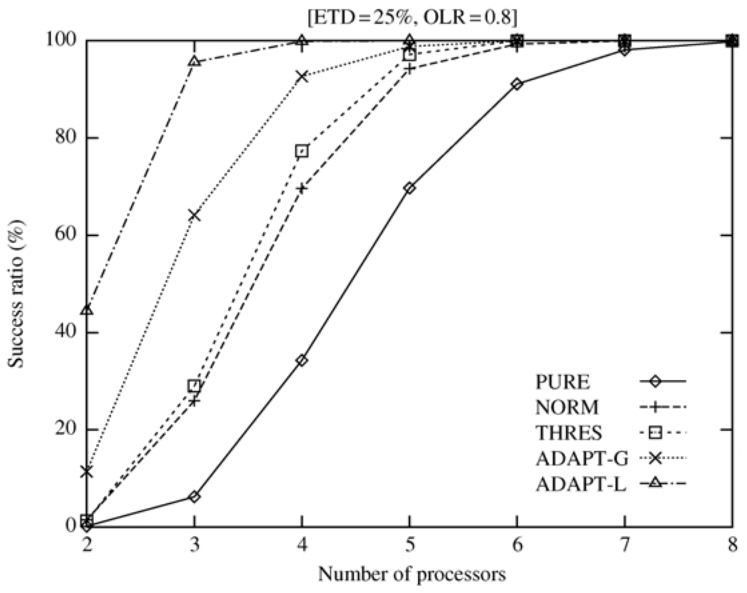

Figure 3. Success ratio as a function of system size.

adaptive metric. This poor performance can be explained by the fact that these metrics do not account for the resource contention that occurs when the application parallelism exceeds the number of available processors. Hence, many task deadlines will be too short to be met. This shortcoming is partly remedied with the ADAPT-G metric. By assigning task laxities based on the knowledge that application parallelism may not be fully exploited on the system, a higher performance can be attained with ADAPT-G. As the plots indicate, this extra intelligence of ADAPT-G gives a significant performance boost for small systems. For example, more than 60% of all scheduling attempts succeed for the three-processor system when ADAPT-G is used. Note that this is more than twice the performance of the best non-adaptive metric.

The ADAPT-L metric exhibits the best performance in this experiment. For example, the plots indicate that around 95% of all scheduling attempts succeed for the three-processor system when ADAPT-L is used. This is a three-fold improvement as compared to the non-adaptive metrics. For a two-processor system, the improvement is even more stunning: the number of successful scheduling attempts is more than an order of magnitude higher for ADAPT-L than for the non-adaptive metrics, and four times higher than for the ADAPT-G metric. The superior performance associated with ADAPT-L can be attributed to the detailed knowledge it possesses regarding the contention that individual tasks will encounter. This is in sharp contrast to ADAPT-G, where the potential contention of each task is derived using the same constant value, namely, the average task graph parallelism. Recall, however, that the superior performance of ADAPT-L comes at the price of a higher time-complexity.

6.2. Effect of Overall Laxity Ratio on the Performance

Another goal with a deadline-distribution strategy is that it should use metrics that perform well for the varying tightness of E-T-E deadline. In particular, by varying the value of OLR when there is significant processor contention, we can identify the capability of each metric to exploit the available surplus time and to distribute it wisely. Figure 4 plots the success ratio as a function of OLR assuming a fixed system size. The plots show some salient results for three processors.

Again, we see that the relative performance of the metrics is the same as in the previous experiment. ADAPT-L still outperforms the other metrics by a significant margin. For example, the performance gain over the non-adaptive metrics is nearly an order of magnitude for tight deadlines. The ADAPT-G metric also performs consistently better than the non-adaptive metrics. For tight deadlines, the plots indicate a three-fold increase in performance for ADAPT-G over the non-adaptive metrics. Also in this experiment, NORM and THRES exhibit comparable performance. However, one can observe an interesting phenomenon from the plots. While, for OLR ≤ 0.9, THRES performs slightly better than NORM, the reverse is true for OLR > 0.9. This effect occurs because of the problem of finding a value of k_S that gives THRES a consistently good performance under varying application and architecture scenarios. This problem will be discussed further in Section 6.6.

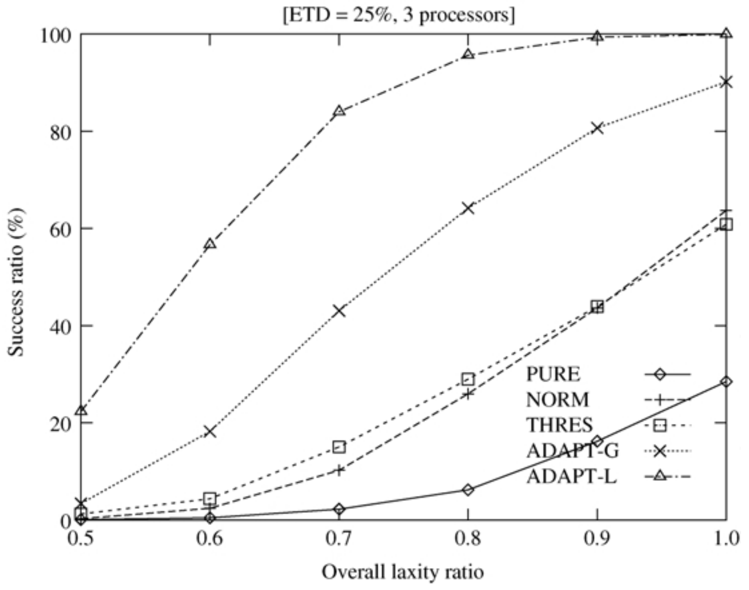

Figure 4. Success ratio as a function of overall laxity ratio.

6.3. *Effect of Execution-Time Distribution on the Performance*

So far, our adaptive metrics have been shown to outperform their non-adaptive counterparts under all system sizes and all degrees of deadline tightness. In this section, we will identify the robustness of each metric under different application scenarios with varying execution-time distributions. Figure 5 plots the success ratio as a function of ETD assuming a fixed system size and a fixed OLR. We show here the results for OLR = 0.8 and three processors while varying ETD from 0% to 100% in steps of 25%.

The observed trends in this experiment are quite similar to the ones demonstrated in the previous experiments, with one notable exception. As the plots clearly indicate, the performance of the NORM metric is not as consistent as in the previous studies. Instead of following the performance of THRES, the performance of NORM will increase past THRES as ETD gets large. This behavior can be attributed to the proportional-share distribution strategy of NORM. With an increasing proportion of short tasks, there will be more tasks with shorter deadlines competing for the processors. NORM is able to compensate long tasks with a larger amount of laxity to handle the contention. The simpler strategy of THRES cannot handle this situation with the same degree of flexibility.

Note that, when ETD = 0%, the PURE, NORM, THRES, and ADAPT-G metrics all converge to the same success ratio (6%), because all tasks have identical execution times c_i. It is easy to see that, with identical execution times, the deadline for each task τ_i in a critical path Φ will be $d_i = D_\Phi / n_\Phi$, regardless of the metric used. In the case of THRES

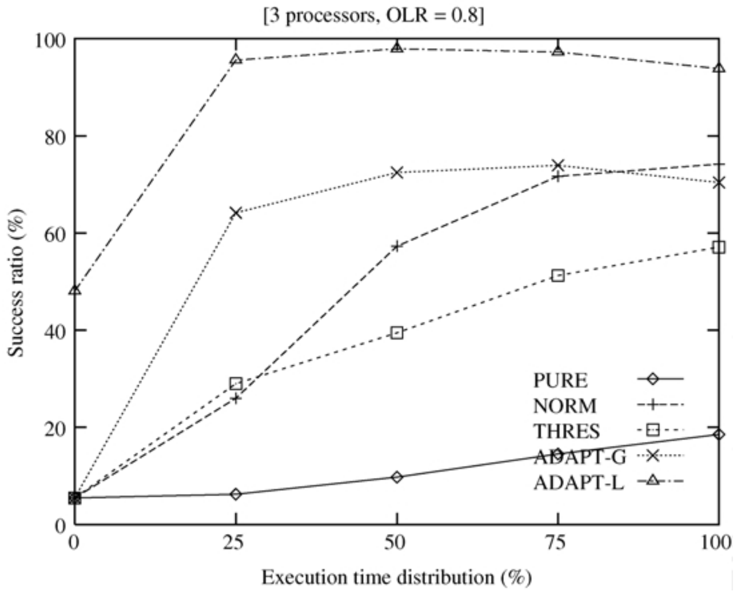

Figure 5. Success ratio as a function of execution time distribution.

or ADAPT-G, the virtual execution times $\hat{c}_i$ are also identical since, for ETD = 0%, the original execution times of all tasks are equal to or higher than the execution time threshold c_{thres} and the surplus factors (k_S and k_G, respectively) are constant. In the case of ADAPT-L, however, the virtual execution time of each task is defined by the size of its parallel set and may thus differ between tasks.

Also, note the anomalous behavior of ADAPT-G and ADAPT-L as ETD exceeds 50%. As we will show in Section 6.6, the performance of the adaptive metrics is most susceptible to changes in ETD. For the chosen values of the adaptivity factors, k_G and k_L, the performance of both ADAPT-G and ADAPT-L drops slightly for applications with a large distribution of execution times.

6.4. Effect of Communication-Cost Estimation Strategy on the Performance

Recall from Section 4.3 that an intuitive approach for communication-cost estimation during deadline assignment in a distributed real-time system is to assume a negligible communication cost between tasks, because (i) many task assignment and scheduling algorithms tend to cluster tasks that communicate heavily, and (ii) in many distributed real-time applications (such as control systems), the communication volume is quite low and therefore any communication cost will be relatively low compared to the task execution times.

In this section, we will show that, even when the conditions (i) or (ii) do not apply, it is still beneficial to assume negligible communication cost. We will investigate three radically different cost estimation strategies. For the Communication cost always assumed (CCAA) strategy, communication channels are modeled as tasks and partake in the deadline distribution with estimated message transmission times used in place of execution times. For the Communication cost never assumed (CCNA) strategy, communication cost is assumed to be zero and deadlines are never distributed over communication channels. For the Communication cost randomly assumed (CCRA), the communication cost estimation strategy is either CCNA or CCAA, depending on the outcome from coin tossing.

Figure 6 plots the success ratio for ADAPT-L as a function of communication-cost estimation strategy assuming a fixed system size. The plotted results are for three processors, and the overall best performance is attained when the CCNA strategy is used. This is because the maximum overall laxity will be available for distribution over the tasks. When the task are subsequently scheduled, any interprocessor communication between tasks will only consume laxity from the receiving task. For the other strategies, precedence constraints in the task graph will consume laxity from the overall laxity pool as these constraints will be modeled as communication tasks with an estimated non-zero communication cost. Each such communication task will thus reduce the amount of laxity available for the tasks in the same path as the communication task. The weakness of this approach is clearly illustrated by the poor performance of the CCRA and CCAA strategies.

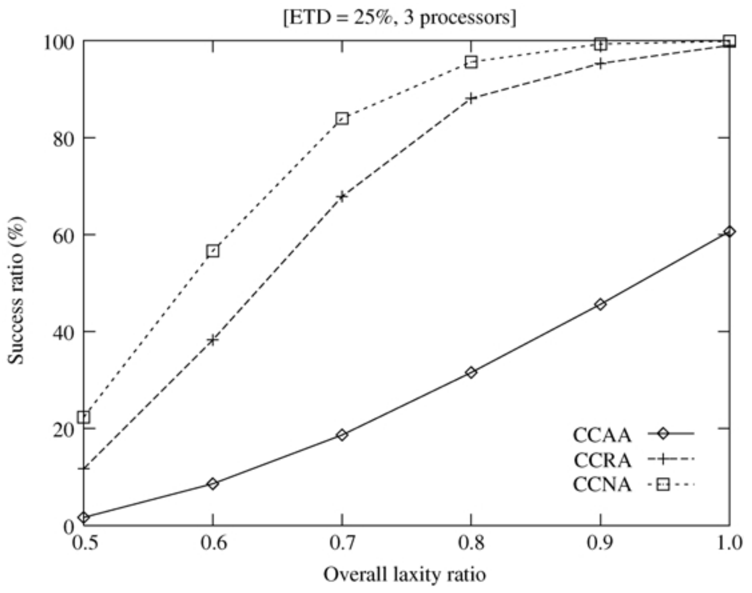

Figure 6. Success ratio for ADAPT-L as a function of communication cost estimation strategy.

6.5. Performance Comparison Using a Secondary Performance Measure

In all studies so far, we have compared the metrics with respect to their contributions to task schedulability. In situations when OLR is large enough, the success ratio will approach 100% for all metrics. In this case, a secondary performance measure should be used to assess the relative quality of the metrics. The maximum task lateness is used for this purpose.

Figure 7 illustrates how the slicing technique behaves for a system with loose E-T-E deadlines (OLR = 1.5). As can be seen in the plots, ADAPT-L clearly outperforms the other metrics for all system sizes. ADAPT-G also performs very well, in particular for small systems. This behavior can also be observed in Figure 8 where the maximum lateness is plotted as a function of ETD. An interesting phenomenon for the NORM metric can be observed in this plot. While the performance of NORM is comparable to that of ADAPT-G for small variations of the task execution time, NORM performs poorly for higher distributions. This behavior can be attributed to the proportional-share strategy employed by NORM. As the proportion of short tasks increases past a certain threshold, the maximum task lateness approaches zero.

6.6. Finding the Best Parameters for the Adaptive Metrics

An interesting question is how to find the best values of the execution time threshold c_{thres} and the factors k_S, k_G, and k_L for a given system. As we will show, this is not an easy task

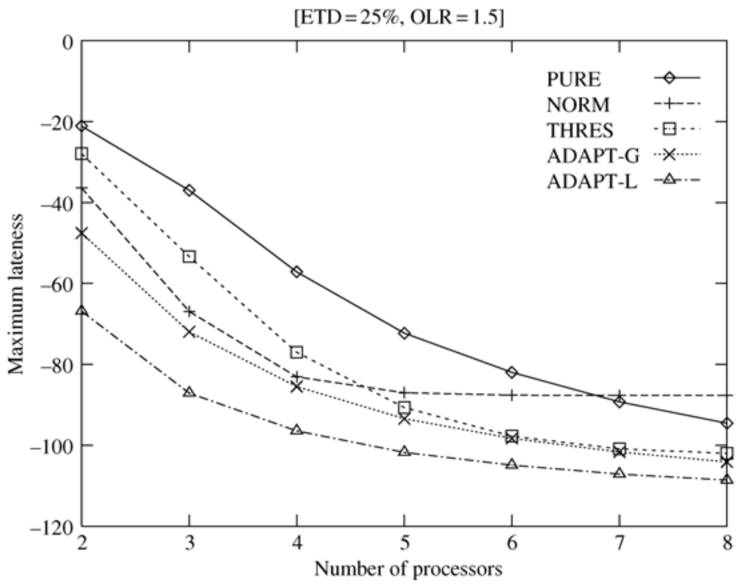

Figure 7. Maximum task lateness as a function of system size.

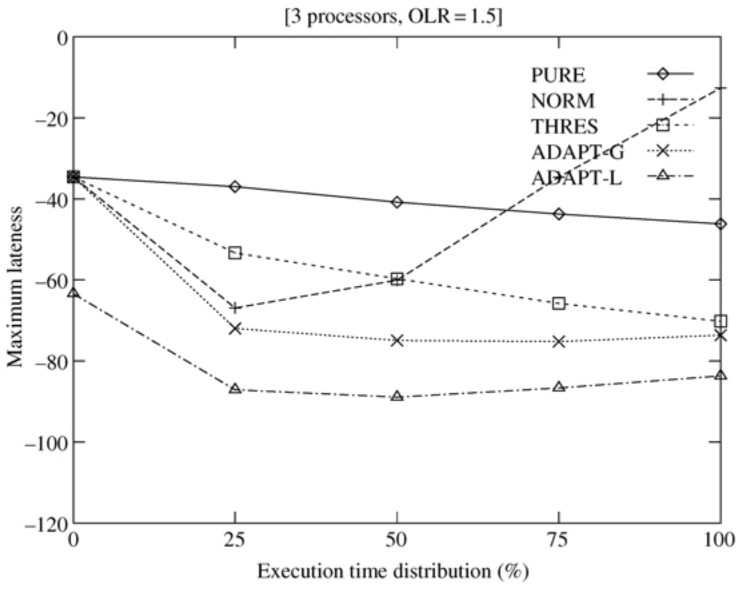

Figure 8. Maximum task lateness as a function of execution time distribution.

owing to the large number of application and architecture attributes that affect the scheduling results.

Figure 9 illustrates how the performance of the THRES metric depends on the value of the surplus factor k_S. The plots show the behavior of THRES as k_S is varied between 0.0 and 4.0 in steps of 0.5. Note that, for $k_S = 0.0$, THRES reduces to PURE. For the default configuration (ETD = 25%, OLR = 0.8), the best performance is attained for $k_S = 2.0$. When ETD is increased, the peak performance is attained at a lower value of k_S. When there is a larger portion of tasks with short execution times, a too high a value of k_S will assign superfluously large amounts of laxity to longer tasks, which will impede the schedulability of shorter tasks. A similar phenomenon can be observed as E-T-E deadlines are made tighter (smaller OLR). With less laxity to distribute for a fixed system size, a smaller value of k_S is preferred in order to allow shorter tasks to be scheduled. When the system size is reduced, an interesting effect kicks in. As the number of processors decreases and full exploitation of application parallelism is no longer possible, it is important to assign extra laxity to larger tasks in order to handle processor contention. Here, the best performance is attained for a higher value of k_S. Because of the counteracting effects that occur when deadlines are made tighter, on the one hand, and system size is decreased, on the other hand, a ''best'' value of k_S is impossible to find for systems which are inflicted with both of these limitations.

Figure 10 illustrates how the performance of the ADAPT-G metric depends on the value of the global adaptivity factor k_G when it is varied between 0.0 and 4.0 in steps of 0.5. Similar to THRES, a lower value of k_G is required for maintained performance as ETD is increased. Also, decreasing the OLR implies lowering k_G. Interesting enough,

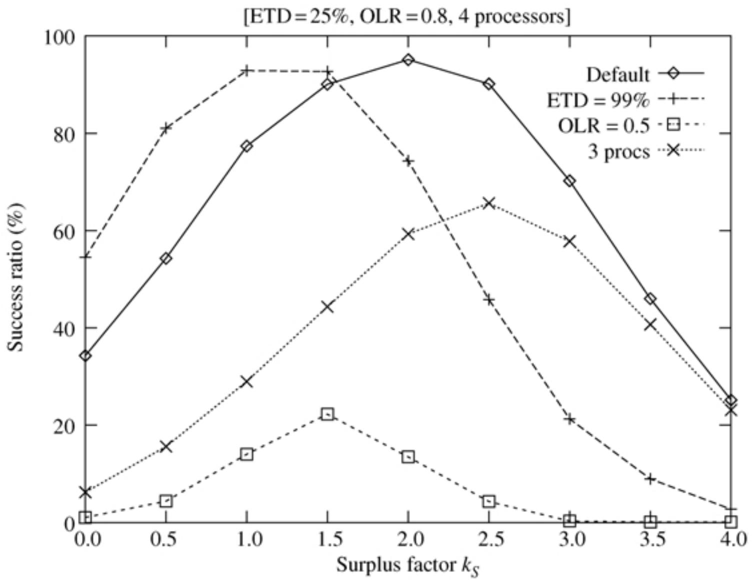

Figure 9. Success ratio for THRES as a function of surplus factor k_S.

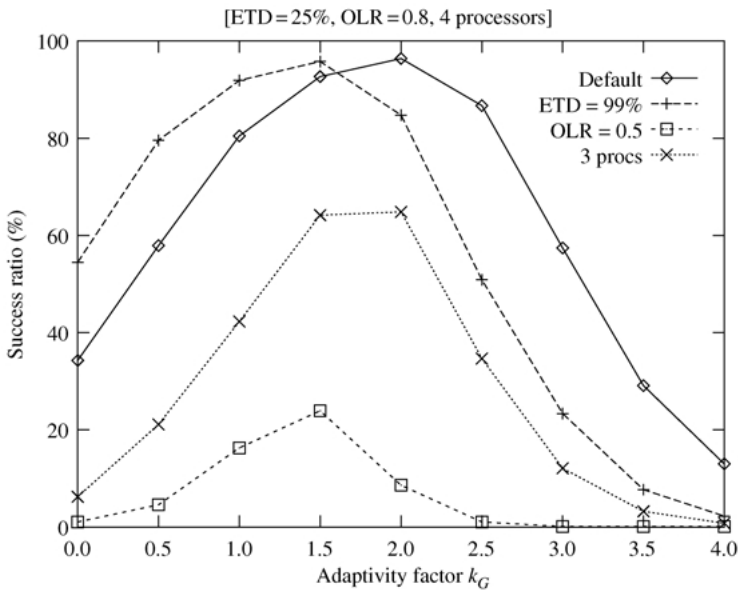

Figure 10. Success ratio for ADAPT-G as a function of global adaptivity factor k_G.

however, decreasing the system size does not require increasing k_G, owing to the adaptive nature of ADAPT-G (see Equation (8)). This means that there exist no conflicting effects that limit the applicability of the metric, as is the case for THRES. A similar behavior for k_L can be observed in the plots for ADAPT-L in Figure 11. Here, k_L is varied between 0.0 and 0.8 in steps of 0.1.

Figure 12 illustrates how the performance of ADAPT-L depends on the value of the execution time threshold c_{thres}. In this figure, c_{thres} is varied between 0.0 and $2.0 * c_{\text{mean}}$ in steps of 0.25. Intuitively, the peak performance should be attained for c_{thres} close to c_{mean}. Too low values of c_{thres} will assign laxities to all tasks and thereby drastically reduce the positive effects of the metric. On the other hand, too high values of c_{thres} will disable the threshold effect and reduce the ADAPT-L metric to the PURE metric, resulting in inferior performance. As can be seen in the plots, the peak performance is attained for $c_{\text{thres}} = c_{\text{mean}}$ for the assumed system. For the given ETD, the performance drops drastically for $c_{\text{thres}} = 1.25 * c_{\text{mean}}$ because, at this point, nearly all tasks are below the threshold. As ETD increases, one can expect that the drastic drop in performance occurs at a higher value of c_{thres} because there will be a larger distribution of task execution times. Again, the plots corroborate this conjecture.

6.7. Summary of the Experiments

In summary, the experiments reported in this section indicate that the proposed adaptive metrics, ADAPT-G and ADAPT-L, outperform (in terms of success ratio and maximum

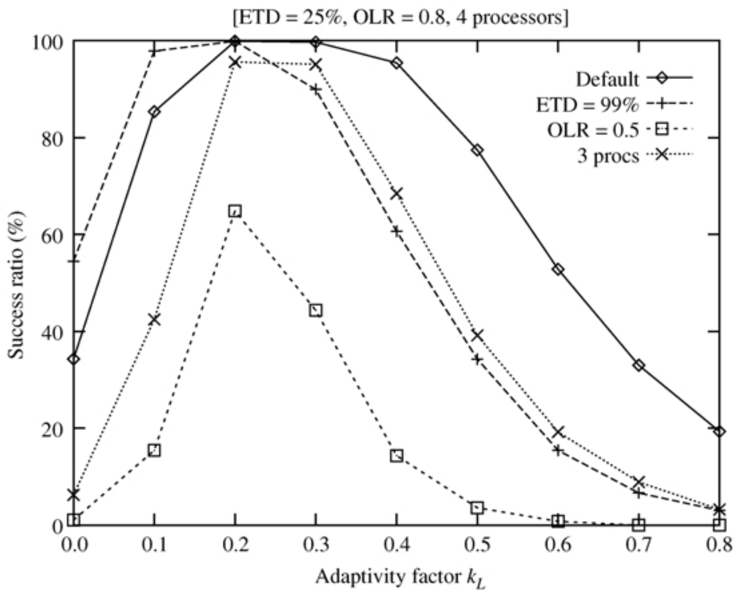

Figure 11. Success ratio for ADAPT-L as a function of local adaptivity factor k_L.

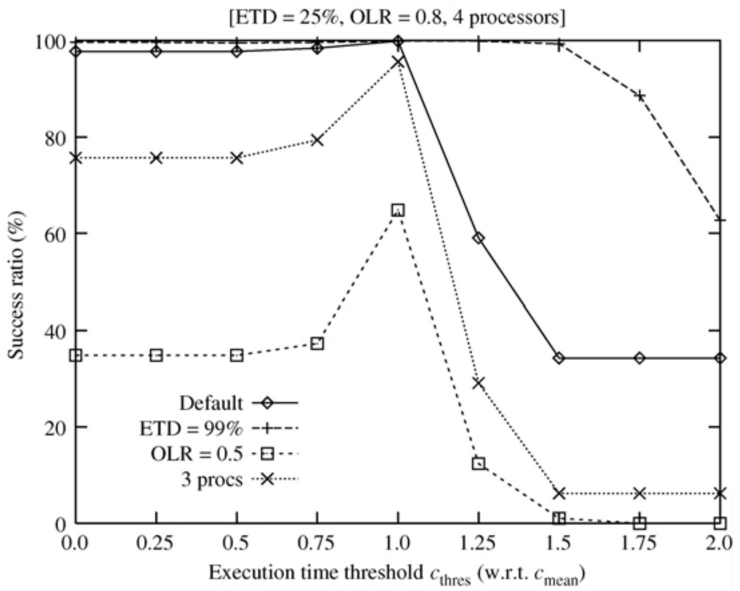

Figure 12. Success ratio for ADAPT-L as a function of execution time threshold c_{thres}.

task lateness) their non-adaptive counterparts for varying system sizes and deadline tightness. In fact, the increase in success ratio can be as high as an order of magnitude for systems with heavy resource contention. Among the adaptive metrics, ADAPT-L is shown to be consistently better than ADAPT-G.

7. Discussion

7.1. Complementary Results

In order to assess the robustness of the adaptive metrics for other parameter settings, we have performed additional experiments. The results from these experiments can be found in Jonsson (1999b), but we briefly summarize them here. We have evaluated the ADAPT-G and ADAPT-L metrics on a heterogeneous processor architecture where a task's execution time may differ depending on which processor the task is assigned to. That study showed that the performance gain (with respect to the success ratio) of the adaptive metrics as compared to their non-adaptive counterparts is maintained when heterogeneity is introduced in the system. We also evaluated the adaptive metrics for task graphs with varying degrees of parallelism and with different probability distributions for the task execution times. Here, we found that the slicing technique scales very well with these parameters. Finally, we evaluated the adaptive metrics for different values of the CCR. Here, too, did we find that the slicing technique scales well with different parameter values. In addition, experiments involving higher values of CCR corroborate the results presented in Section 6.4, namely, that the best strategy is to completely ignore the interprocessor communication costs during deadline assignment.

We have also assessed the performance of the adaptive metrics on a more absolute scale by investigating whether the generated task sets are difficult to schedule to begin with. To this end, we used an optimal combined deadline-distribution and task-assignment algorithm implemented by means of the parametrized branch-and-bound (B&B) framework available in the GAST evaluation tool (Jonsson, 1998, 1999a). This experiment revealed that the ADAPT-L metric in fact delivered a success ratio that was no more than 10 percentage points[8] worse than that of the optimal B&B algorithm on the average. This indicates that the ADAPT-L metric is so powerful that no further improvements of the technique are needed. This also indicates that the performance of other assignment and scheduling optimization techniques, such as simulated annealing (Tindell et al., 1992), can be expected to perform only slightly better for similar types of applications, but with an equal or higher time complexity.

7.2. Practicality Issues

For the practical applicability of the adaptive metrics (ADAPT-G, ADAPT-L), it is important to find the values of the factors k_G and k_L for a given application. One must note that there exists no overall best value of any factor. In Section 6.6, the values used in our experiments were found to be the best for the generated random task graphs. We believe that these values will also be useful for many other graphs for the following reason. According to Equations (8) and (10), the adaptivity factors of ADAPT-G and ADAPT-L are clearly sensitive to the actual application parallelism. However, as long as the parallelism in the task graph deviates within reasonable bounds from the values chosen in our experiments, the adaptive metrics will still outperform the non-adaptive counterparts. Minor adjustments of the factors will probably be necessary for other application task sets, but, as long as the slicing technique is not applied to strictly sequential or massively parallel applications, robust factor values should not be too hard to find.

The question of whether to use ADAPT-G or ADAPT-L depends, to a large extent, on the nature of the target system. For example, in a system where the characteristics of all tasks are known before the system is put in use, off-line scheduling is used. In this case, the cost of the deadline-distribution algorithm is often not a major concern and hence the ADAPT-L metric could be used. For a system with on-line scheduling, on the other hand, tasks typically arrive dynamically and hence the scheduling complexity is a major source of concern since the scheduling process should not consume too much time during run-time task dispatching. In such a case, ADAPT-G may be a better choice owing to its lower complexity.

The choice of metric will also be affected by the underlying task assignment and scheduling strategy. If a greedy scheduling algorithm, typically with a computational complexity of $O(n^2)$ or lower, is used, applying the ADAPT-G metric adds little to the total complexity. Applying ADAPT-L in this situation could make the complexity prohibitively high. However, if a backtracking or branch-and-bound algorithm is used, the high complexity of these algorithms would make the ADAPT-L metric a viable alternative as its $O(n^3)$ complexity should be negligible in comparison.

Finally, a comment of the complexity of the ADAPT-L metric. One problem with the original definition of the parallel set Ψ_i in Equation (10) is that it may include tasks that cannot possibly contend with τ_i, namely, those that have been assigned slices in a previous iteration of the deadline-distribution algorithm. In order to more accurately assess the potential resource contention for each task, we have, therefore, also evaluated the effect of a parallel set $\hat{\Psi}_i$ that only includes tasks in the original parallel set Ψ_i whose execution windows may overlap with any feasible execution window of τ_i. In this context, two execution windows w_i and w_j are said to overlap if $a_i \leq D_j \wedge a_j \leq D_i$. Unfortunately, with the definition of the parallel set $\hat{\Psi}_i$ comes the drawback that it has to be recalculated each time a new critical path is found. A recalculation of $\hat{\Psi}_i$ for all tasks (given an already existing $\hat{\Psi}_i$) can be done in time $O(n^2)$, so the total time-complexity of the main loop of the deadline-distribution algorithm in Figure 1 would increase to $O(n^3)$. When the slicing technique was used with the parallel set $\hat{\Psi}_i$, only a small performance increase ($< 2\%$) was observed. Given the already high gain in performance, this small additional increase in performance normally do not warrant the use of $\hat{\Psi}_i$ with its significant contribution to the algorithm complexity. In line with the above discussion, the choice of whether to use this improvement or not is again a function of the actual system configuration.

7.3. Future Work

The techniques presented here can be applied not only to computational resources such as processors, but also to general resources including shared data structures and communication links. Future work will therefore include evaluation of techniques that can also include such general resource requirements. In particular, we would like to further evaluate the strategy of modeling interprocessor communication as separate tasks which contribute to the deadline distribution. Such an evaluation could indicate at what interprocessor communication volumes it is more beneficial to use that modeling strategy instead of the one advocated in this work where communication is not included in deadline distribution.

In this work we have mainly studied randomly-generated task graphs. The results attained from this study are therefore merely indicative, rather than evidential, of the performance of the slicing technique with practical applications. To be able to make more general statements on the merits of our proposed approach, we would like to evaluate the slicing technique on a set of realistic benchmarks that encompass small comprehensible applications as well as large applications.

8. Conclusions

Distribution of end-to-end deadlines over component tasks in a distributed real-time application is an important, but difficult, problem to solve. It is particularly difficult for those systems with relaxed locality constraints where a majority of tasks are not pre-assigned to particular processors. Many solutions have been proposed for the problem, but mostly for systems with strict locality constraints where task assignment is entirely known beforehand.

In this paper we have presented two adaptive metrics for an existing deadline-distribution algorithm. The first metric, ADAPT-G, accounts for average application parallelism and system size so as to increase the likelihood of successfully finding a feasible schedule. The time-complexity of the algorithm when using ADAPT-G is $O(n^2)$, which is comparable to that of previously-proposed non-adaptive metrics. The second metric, ADAPT-L, utilizes information about possible resource contention for individual tasks in the application. The time complexity of the algorithm when using ADAPT-L is $O(n^3)$. The results from an extensive simulation study show that the proposed metrics outperform (in terms of success ratio and maximum task lateness) their non-adaptive counterparts over a wide range of system configurations. In particular, we find that, for small systems, ADAPT-L outperforms the non-adaptive metrics with as much as an order of magnitude with respect to the success ratio. Furthermore, ADAPT-L outperforms the ADAPT-G metric with a three-fold increase in performance. Overall, the new adaptive metrics are found to exhibit very good performance over a large variety of application and architecture scenarios.

Acknowledgments

The authors thank the anonymous referees for their comments which helped improve the quality of this paper. This research was supported in part by a grant from the Volvo Research Foundation and the Volvo Educational Foundation.

Notes

1. By ''execution-time distribution'' we mean the variance in task execution time, taken over all tasks in the application; not the variance of a single task's execution time as are typically used in stochastic real-time task models.
2. Although the slicing technique has been evaluated under a non-preemptive, time-driven scheduling policy, it is not restricted to that policy as we showed by property P1 in Section 2.1.
3. We assume that all related tasks in one application have the same periods, which is reasonable.
4. Note that, through the introduction of the planning cycle, the periodicity assumption for the tasks has been relaxed. This means that our system model can easily be extended to the case where tasks are not periodic.
5. It should be noted that many other real-time systems (e.g., multimedia or remote sensing systems) may experience significant intertask communication. In these cases, it might be a better choice to model the communication as separate tasks and include them into the distribution of the E-T-E deadline.
6. The conclusions drawn from the experiments are not dependent on a shared-bus topology or a constant per-item communication cost. In fact, the results apply for any communication network as long as an upper-bounded communication cost can be derived for each pair of communicating tasks. Hence, we can also use networks such as CAN and Token Ring.
7. If the scheduling of network messages is performed in close cooperation with task scheduling (similar to the techniques in (Abdelzaher and Shin, 1995; Tindell et al., 1992)), it is possible to use more accurate models of the communication network. This would result in tighter, and less pessimistic, communication-cost estimates, which of course would improve the performance of our proposed approach.
8. The maximum value of 10 percentage point was obtained for the worst-possible application constraints, for example, very tight deadlines or a low number of processors. For more relaxed application constraints, the ADAPT-L metric delivered performance that was comparable (within 3 percentage points) to that of the optimal algorithm.

References

Abdelzaher, T. F., and Shin, K. G. 1995. Optimal combined task and message scheduling in distributed real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*. Pisa, Italy, pp. 162–171. An improved version also appeared in *IEEE Transactions on Parallel and Distributed Systems* 10(11): 1179–1191.

Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. J. 1993. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal* 8(5): 284–292.

Bettati, R., and Liu, J. W.-S. 1992. End-to-end scheduling to meet deadlines in distributed systems. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*. Yokohama, Japan, pp. 452–459.

Cheng, S., Stankovic, J. A., and Ramamritham, K. 1986. Dynamic scheduling of groups of tasks with precedence constraints in distributed hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*. New Orleans, Louisiana, pp. 166–174.

Chetto, H., Silly, M., and Bouchentouf, T. 1990. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems* 2(3): 181–194.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. 1990. *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press.

Di Natale, M., and Stankovic, J. A. 1994. Dynamic end-to-end guarantees in distributed real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*. San Juan, Puerto Rico, pp. 216–227.

Di Natale, M., and Stankovic, J. A. 1995. Applicability of simulated annealing methods to real-time scheduling and jitter control. In *Proceedings of the IEEE Real-Time Systems Symposium*. Pisa, Italy, pp. 190–199.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman.

Garey, M. R., Johnson, D. S., Simons, B. B., and Tarjan, R. E. 1981. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing* 10(2): 256–269.

Gutiérrez García, J. J., and González Harbour, M. 1995. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *Proceedings of the IEEE Workshop on Parallel and Distributed Real-Time Systems*. Santa Barbara, California, pp. 124–132.

Hou, C.-J., and Shin, K. G. 1997. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Transactions on Computers* 46(12): 1338–1356.

Jonsson, J. 1997. The impact of application and architecture properties on real-time multiprocessor scheduling. Ph.D. thesis, School of Electrical and Computer Engineering, Chalmers University of Technology, Göteborg, Sweden.

Jonsson, J. 1998. GAST: a flexible and extensible tool for evaluating multiprocessor assignment and scheduling techniques. In *Proceedings of the International Conference on Parallel Processing*. Minneapolis, Minnesota, pp. 441–450.

Jonsson, J. 1999a. Effective complexity reduction for optimal scheduling of distributed real-time applications. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*. Austin, Texas, pp. 360–369.

Jonsson, J. 1999b. A robust adaptive metric for deadline assignment in heterogeneous distributed real-time systems. In *Proceedings of the IEEE International Parallel Processing Symposium*. San Juan, Puerto Rico, pp. 678–687.

Kao, B., and Garcia-Molina, H. 1993. Deadline assignment in a distributed soft real-time system. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*. Pittsburgh, Pennsylvania, pp. 428–437.

Kao, B., and Garcia-Molina, H. 1994. Subtask deadline assignment for complex distributed soft real-time tasks. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*. Poznan, Poland, pp. 172–181.

Kopetz, H. 1992. Sparse time versus dense time in distributed real-time systems. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*. Yokohama, Japan, pp. 460–467.

Leung, J. Y.-T., and Merrill, M. L. 1980. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters* 11(3): 115–118.

Peng, D.-T., and Shin, K. G. 1989. Static allocation of periodic tasks with precedence constraints in distributed real-time systems. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*. New Port Beach, California, pp. 190–198.

Raji, R. S. 1994. Smart networks for control. *IEEE Spectrum* pp. 49–55.

Ramamritham, K. 1995. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems* 6(4): 412–420.

Rexford, J. L., Hall, J., and Shin, K. G. 1996. A router architecture for real-time point-to-point networks. In *Proceedings of the ACM International Symposium on Computer Architecture*. Philadelphia, Pennsylvania, pp. 237–246.

Saksena, M., and Hong, S. 1996a. An engineering approach to decomposing end-to-end delays on a distributed real-time system. In *Proceedings of the IEEE Workshop on Parallel and Distributed Real-Time Systems*. Honolulu, Hawaii, pp. 244–251.

Saksena, M., and Hong, S. 1996b. Resource conscious design of distributed real-time systems: an end-to-end approach. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems*. Montreal, Canada, pp. 306–313.

Tindell, K. W., Burns, A., and Wellings, A. J. 1992. Allocating hard real-time tasks: an NP-hard problem made easy. *Real-Time Systems* 4(2): 145–165.

Wang, F., Ramamritham, K., and Stankovic, J. A. 1992. Bounds on the performance of heuristic algorithms for multiprocessor scheduling of hard real-time tasks. In *Proceedings of the IEEE Real-Time Systems Symposium*. Phoenix, Arizona, pp. 136–145.

Jan Jonsson received the M.S. degree in computer science and engineering, the Lic.Tech. degree in computer engineering and the Ph.D. degree in computer engineering, all from Chalmers University of Technology, Sweden, in 1992, 1995, and 1997, respectively. He is currently an associate professor in the Department of Computer Engineering at Chalmers University of Technology. During fall/winter 1996 he was a visiting researcher at the Real-Time Computing Laboratory, The University of Michigan, Ann Arbor, USA.

He has authored/co-authored numerous technical papers within the field of real-time scheduling in multiprocessor architectures, a dozen of which have been presented at competitive, fully-reviewed international conferences and workshops. His research interests are in the areas of parallel and distributed real-time computing and multiprocessor computer architecture. He is a member of the IEEE, IEEE Computer Society, ACM, and ACM SIGARCH.

Kang G. Shin is Professor and Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan.

He has supervised the completion of 40 Ph.D. theses, and authored/co-authored over 600 technical papers and numerous book chapters in the areas of distributed real-time computing and control, computer networking, fault-tolerant computing, and intelligent manufacturing. He has co-authored (jointly with C. M. Krishna) a textbook *Real-Time Systems,* McGraw Hill, 1997. In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award, and Research Excellence Award in 1989, Outstanding Achievement Award in 1999, and Service Excellence Award in 2000 from The University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are investigating various issues related to real-time and fault-tolerant computing.

His current research focuses on Quality of Service (QoS) sensitive computing and networking with emphasize on timeliness and dependability. He has also been applying the basic research results to telecommunication and multimedia systems, intelligent transportation systems, embedded systems, and manufacturing applications.

He received the B.S. degree in Electronics Engineering from Seoul National University, Seoul, Korea in 1970, and both the M.S. and Ph.D. degrees in Electrical Engineering from Cornell University, Ithaca, New York in 1976 and 1978, respectively. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the US Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at UC Berkeley, and International Computer Science Institute, Berkeley, CA, IBM T. J. Watson Research Center, and Software Engineering Institute at Carnegie Mellon University. He also chaired the Computer Science and Engineering Division, EECS Department, The University of Michigan for three years beginning January 1991.

He is an IEEE fellow and member of the Korean Academy of Engineering, is the General Chair of the 2000 IEEE Real-Time Technlogy and Applications Symposium, was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 August special issue of *IEEE Transactions on Computers* on Real-Time Systems, a Program Co-Chair for the 1992 *International Conference on Parallel Processing*, and served numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991–93, was a Distinguished Visitor of the Computer Society of the IEEE, an Editor of *IEEE Transactions on Parallel and Distributed Computing*, and an Area Editor of *International Journal of Time-Critical Computing Systems* and *Computer Networks*.