Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems

Chao-Ju Hou, *Member*, *IEEE*, and Kang G. Shin, *Fellow*, *IEEE*

Abstract—This paper addresses the problem of allocating (assigning and scheduling) periodic task modules to processing nodes in distributed real-time systems subject to task precedence and timing constraints. Using the branch-and-bound technique, a module allocation scheme is proposed to find an "optimal" allocation that maximizes the probability of meeting task deadlines.

The task system within a planning cycle is first modeled with a task flow graph which describes computation and communication modules, as well as the precedence constraints among them. To incorporate both timing and logical correctness into module allocation, the probability of meeting task deadlines is used as the objective function. The module allocation scheme is then applied to find an optimal allocation of task modules in a distributed system. The timing aspects embedded in the objective function drive the scheme not only to assign task modules to processing nodes, but also to use a module scheduling algorithm (with polynomial time complexity) for scheduling all modules assigned to each node, so that all tasks may be completed in time.

In order to speed up the branch-and-bound process and to reduce the computational complexity, a dominance relation is derived from the requirement of timely completion of tasks and use to eliminate the possibility of generating vertices in the state-space search tree, which never lead to an optimal solution, and an upper bound of the objective function is derived for every partial allocation with which the scheme determines whether or not to prune the corresponding intermediate vertex in the search tree. Several numerical examples are presented to demonstrate the effectiveness and practicality of the proposed scheme.

Index Terms—Real-time systems, dynamic failure, task/module allocation, module scheduling, precedence and deadline constraints, task flow graph, branch-and-bound process.

1 Introduction

The availability of inexpensive, high-performance processors and high-capacity memory chips has made it attractive to use distributed computing systems for real-time applications. For example, one can make the execution of both periodic and aperiodic tasks not only logically correct, but also completed before their deadlines, by partitioning periodic tasks into a set of communicating modules, statically allocating these modules to processing nodes (PNs) in a distributed system, and dynamically distributing aperiodic tasks as they arrive according to the load state of each PN.

Partitioning tasks are usually based on some application-dependent criterion and the system architecture under consideration, while the dynamic distribution of aperiodic tasks is usually treated as an adaptive *load sharing* problem. Both of these are not the intent of this paper; see [1] for an example of partitioning real-time tasks into modules/activities, and see [2], [3], [4], [5], [6] for examples of dynamic load sharing in distributed real-time systems. In this paper, we consider, instead, the issue of "optimally" (in the sense to be defined later) allocating periodic task modules to PNs in a distributed system, so as to fully utilize the inherent parallelism, modularity, and reliability of the system, while alleviating the "saturation effect" [7] caused by excessive interprocessor communication of data and control messages.

The problem of allocating tasks/modules in a distributed system has been studied by many researchers with respect to different objective functions. These objective functions can be roughly grouped into four categories:

1) Minimization of total computation and communication times in the system [7], [8], [9], [10]. In the case of homogeneous systems, this objective function reduces to the minimization of the total interprocessor communication time.
2) Load balancing by minimizing the statistical variance of processor utilization [11], [12] or by maximizing the total rewards in the semi-Markov process that models the computer system [13].
3) Minimization of maximum computation and communication times on a PN, the objective function of which was termed the *maximum turnaround time* in [14], the bottleneck processor time in [15], [16], and the system hazard in [17].
4) Maximization of the reliability function of both PNs and communication links [18], [19].

Different objective functions lead to different optimality conditions and different allocation results. The first two

- C.-J. Hou is with the Department of Electrical Engineering, The Ohio State University, Columbus, OH 43210-1272. E-mail: jhou@ee.eng.ohio-state.edu.
- K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122. E-mail: kgshin@eecs.umich.edu.

Manuscript received 1 Mar. 1995; revised 13 Aug. 1997.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105672.

objectives are suitable for a distributed system executing multiple simultaneous non-real-time applications, where maximizing the total throughput or minimizing the average response time is the main concern. However, for real-time systems, the timing and logical correctness of each individual task must be considered, because failure to correctly complete a task in time could cause catastrophe. Thus, the third objective function, which is based on the worst-case behavior, is more suitable for assessing the timeliness of real-time systems, while the fourth objective function, that incorporates reliability into task/module allocation, is more suitable for assessing logical correctness. In this paper, we use the probability of completing each task with *both* timing and logical correctness as the objective function, which was termed, in [20], the *probability of no dynamic failure* (P_{ND}). Specifically, P_{ND} is the product of two component probabilities:

- The probability, P_{ND1}, that all tasks within a *planning cycle* are completed before their deadline. The planning cycle is the time period within which the task-invocation behavior repeats itself throughout the entire mission, and, thus, completely specifies the entire task system. More on this will be discussed in Section 2.
- The probability, P_{ND2}, that all PNs are operational during the execution of task modules assigned to them, and the links between communicating PNs[1] are operational for all intermodule communications over these links under a given allocation.

Consequently, the use of P_{ND} incorporates *both* timeliness and logical correctness into task allocation. This is in sharp contrast to the other module allocation approaches reported in the literature, which deal with either average task response time or logical correctness, but not both.

The problem of finding an optimal assignment of tasks/modules to processors subject to precedence constraints has been shown to be NP–hard for all the above problem formulations [21], [22], and some form of enumerative optimization and/or local search approaches must be sought. In this paper, we develop a module allocation (**MA**) scheme using the branch–and–bound (**BB**) method. We first model the task system with a task flow graph (TG), which describes computation and communication modules as well as the precedence constraints among them. We then use the **BB** method to search for an optimal module allocation. The computational complexity is reduced by deriving an upper bound of the objective function with which we determine whether to expand or prune intermediate vertices (corresponding to partial allocations) in the state-space search tree. On the other hand, because of the timing aspects embedded in the objective function, the performance of any resulting assignment strongly depends on how the assigned tasks/modules are scheduled. Thus, when evaluating an upper-bound (exact) objective function for a partial (complete) allocation, we use a module scheduling (**MS**) algorithm (with polynomial time complexity) to schedule all the modules assigned to a PN by minimizing the maximum tardiness of modules subject to precedence constraints. The **MA** scheme, combined with the **MS** algorithm, is guaranteed to find the optimal allocation of modules to PNs subject to precedence constraints. By "allocation," we mean the *assignment* of modules coupled with the *scheduling* of all modules assigned to each PN.

Shen and Tsai [14] minimized the maximum turnaround time, but considered only a single invocation of each task. They did not take into account precedence constraints between tasks. Chu et al. [15], [16] chose to minimize the bottleneck processor workload for tasks/modules assignment, but their algorithm/analysis was solely based on *mean* task response times, which eliminates the need to consider the scheduling problem. Peng and Shin [17] are the first to include the important timing aspects in the objective function, and combine task scheduling with task assignment. They chose to minimize the *system hazard*, which is defined as the maximum normalized (with respect to task period) task flowtime. It is, however, not clear how system hazard is related to the probability of no dynamic failure. The restriction on assigning all modules of the same task to a single PN may not always be desirable.

Ramamritham [23] used a heuristic-directed search technique with tunable design parameters to

1) determine whether or not a group of communicating modules should be assigned to the same PN, and
2) allocate different groups of modules to PNs and schedule them with respect to their latest–start–times and precedence constraints.

Compared to this work, we use a finer granularity in modeling the real-time task system. (For example, we include probabilistic branches/loops in task graphs and allow communications between periodic tasks.) Also, no conclusions were made on whether or not his algorithm always leads to an optimal solution. By contrast, we focus on module allocation subject to precedence and timing constraints, as well as on the minimization of P_{ND}, which, as mentioned earlier, takes into account both timeliness and reliability. Also, as will be demonstrated in our simulation study later, the **MA** scheme always finds the best allocation at tractable computational costs for task systems with less than 50 modules and/or distributed systems with less than 40 PNs.

The rest of the paper is organized as follows. In Section 2, we discuss how to model real-time task systems. Assumptions on the distributed system are also stated there. In Section 3, we provide an overview of our module allocation scheme. The objective functions, $P_{ND1}(x)$ and $P_{ND2}(x)$—with which an allocation x is assessed in terms of timeliness and logical correctness—are derived in Sections 4 and 5. In Section 6, we address how to achieve both branching and bounding efficiencies. Section 7 presents demonstrative examples. The paper concludes with Section 8.

1. By communicating PNs, we mean a pair of PNs to which two communicating modules are assigned under a given allocation.

2 Task and System Models

2.1 The Task System

Real-time tasks are either periodic or nonperiodic. A periodic task is invoked at fixed time intervals and constitutes the base load of the system. Its attributes, such as the required resources, the execution time, and the invocation period, are usually known a priori. A nonperiodic task, on the other hand, is invoked randomly in response to environmental stimuli, especially to unanticipated abnormal situations. The main intent of this paper is to address the problem of allocating the modules of periodic tasks.

2.1.1 Planning Cycle

To analyze the behavior of periodic tasks, we only need to consider the task behaviors within a specific period, the task behaviors during which will repeat for the entire mission lifetime. Such a period is called the *planning cycle* of periodic tasks and is defined as the least common multiple (LCM) L of $\{p_i : i = 1, 2, ..., N_T\}$, where p_i is the period of a task T_i and N_T is the total number of periodic tasks in the system. That is, the planning cycle is the time interval $[t_0 + kL, t_0 + (k + 1)L)$, where t_0 is the mission start time, and k is a nonnegative integer.

2.1.2 Attributes and Precedence Constraints Among Modules

Each task can be decomposed into smaller units, called *modules* [1]. Each module M_i requires e_i units of execution time. The execution time of a module could be its worst-case execution time or its real execution time, if known. Since extensive simulations and testing are required before putting any critical real-time system in operation (e.g., fly-by-wire computers), the system designer is assumed to have a good, albeit sometimes incomplete, understanding of either the exact execution time or the worst-case execution time of each module.

The execution order of modules imposes precedence constraints among them. These precedence constraints are of the form $M_i \rightarrow M_j$, meaning that the completion of M_i of a task enables another module M_j of the same task to be ready for execution (e.g., by letting M_i send a short message to enable M_j's execution and/or update the data variables/files shared between them [15], [16]). On the other hand, tasks communicate with one another to accomplish the overall control mission. The semantics of message communication between two cooperating tasks also impose precedence constraints between the associated modules of these tasks. This kind of precedence constraint is also of the form $M_i \rightarrow M_j$, except that M_i and M_j now belong to different tasks.

If M_i and M_j are assigned to the same PN, communication between them can be achieved via accessing shared memory. Overheads of such communications are usually much smaller than those when M_i and M_j reside on different PNs. Any two communicating modules that reside on two different PNs will incur interprocessor communication (IPC). IPC introduces a communication delay, which is a function of intermodule communication (IMC) volume (measured in data units) and the *worst case link delay* (or delay per data unit) between the two communicating PNs.[2]

It is important to observe that, even if exact *module* execution times were known in advance, *task* execution times are not known, due to, for instance, the existence of data-dependent (thus, probabilistic) branches/loops in the task (to be discussed below) and/or inexact knowledge of IMC/IPC delays.

2.1.3 Task Flow Graph (TG)

A TG is commonly used to describe the logical structure of modules, and the communications and precedence constraints among them. A TG is composed of four types of subgraphs: chain, AND-subgraph, OR-subgraph, and loop. See [1], [24] for a detailed account of the four component subgraphs. Here we assume that the probability for taking a particular branch in an OR-subgraph or for repeating/exiting the body of a loop is assumed to be independent of that for others (residing either in modules of different tasks or in modules of the same task). These probability values could be set to worst-case values and can be obtained from the extensive simulations and testing—usually required of critical real-time systems—during the system-design phase. We also assume that the number of times a loop can be executed is no more than its maximum loop count. Imposing a maximum loop count for each loop is necessary for real–time applications, since each real-time task must be completed in a finite time. Fig. 1a shows a simple example of a TG.

2.1.4 Communication Primitives

The semantics of the most general communication primitive SEND-RECEIVE-REPLY,[3] can be embedded into precedence relations between modules. If module M_a of task T_i issues a SEND to task T_j, T_i remains blocked, or cannot execute module M_b that follows M_a until the corresponding REPLY from T_j is received. If the module, M_c, responsible for the corresponding communication activity on T_j's side executes a RECEIVE before the SEND arrives, T_j also remains blocked. For example, the communication activities between tasks in Fig. 1a can be embedded into the precedence constraints between modules, as shown in Fig. 1b.

2.2 The Distributed System

The distributed system considered here consists of K processing nodes (PNs). For clarity of exposition, all PNs are assumed to have the same processing power and the same set of resources. (This assumption can, however, be easily relaxed, but with more complex notation.) The time required by an IMC within a PN is assumed to be negligible, while that between two PNs is expressed as the product of the IMC volume (measured in data units) and the *worst case* link delay (measured in time units per data unit) between the two PNs on which the communicating modules reside.[4] Note that the worst case link delay is the communication delay bound guaranteed by the underlying communication subsystem to provide to messages with time constraints. Here, we

2. See Section 2.2 for the reason of expressing IPC as a function of IMC volume and worst case link delay.

3. Other communication primitives, such as QUERY–RESPONSE and WAITFOR [1], can always be realized using SEND–RECEIVE–REPLY.

4. The time for packetization and depacketization is lumped into module execution time for the clarity of algorithm description.

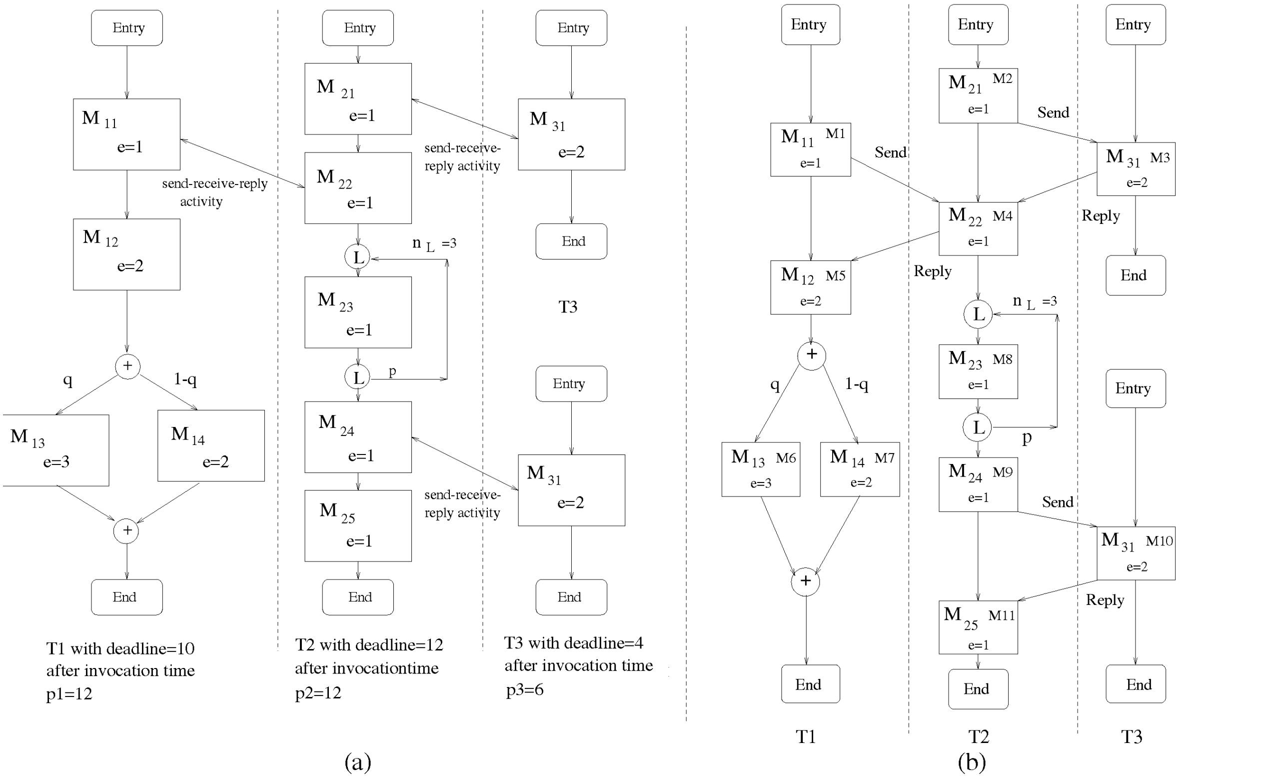


Fig. 1. An example of task flow graph. The label that appears in the upper right corner of each box in (b) is the acyclic number associated with each module.

assume that the communication subsystem and the underlying protocol support time-constrained communications, and the worst-case delay experienced by time-constrained messages is bounded and predictable. Examples of such communication subsystems are the highly responsive token ring described in [25], the FDDI network with the timed-token MAC protocol [26], [27], [28], [29], [30], the distributed queue dual bus (DQDB) network with the isochronous services [31], [32], and the point–to–point packet-switched network described in [33], [34]. No restriction is imposed on the topology of the communication subsystem.

Each PN N_k and each link ℓ_{mn} between N_m and N_n are assumed to fail independently with exponential rates λ_k and $\hat{\lambda}_{mn}$, respectively. We do not take into account of repair and recovery times for failed PNs in assessing the logical correctness of an allocation. Instead, we shall allocate modules to PNs on which failures are least likely to occur during the execution of task modules. That is, we shall assign modules to PNs with maximum reliability and thus eliminate the need of on-line repair and recovery. To guard against the unlikely failures of these PNs, one can assign copies of a module to multiple PNs, but this subject is not the scope of this paper. The rationale behind the above assumption is that

1) repair and recovery times are largely implementation-dependent, and
2) repair and recovery routines usually introduce too high time overheads to be used on-line for time-critical applications.

3 Module Allocation Algorithm

Let N be the number of modules to be allocated within a planning cycle. The module allocation problem can be formulated as that of maximizing $P_{ND}(x) = P_{ND1}(x) \cdot P_{ND2}(x)$ over all possible allocations subject to

$$\sum_{k=1}^{K} x_{ik} = 1, \quad \text{for } 1 \le i \le N,$$

where $x_{ik} = 1$ if and only if M_i is assigned to N_k, $P_{ND1}(x)$ is the probability that all tasks meet their deadlines under allocation x, and $P_{ND2}(x)$ is the probability that all PNs are operational during the execution of modules assigned to them and all communication links are operational during the IPCs that use these links under x. As will be clear later, the precedence constraints among modules are figured in the calculation of module *release times* (to be defined later), and the timing constraints on modules/tasks are considered when $P_{ND1}(x)$ is evaluated; for example, $P_{ND1}(x) = 0$ if some task in the TG misses its deadline under x. The expressions for $P_{ND1}(x)$ and $P_{ND2}(x)$ will be derived in Sections 4 and 5.

To solve the above module allocation problem, we propose the **MA** scheme which uses:

Branch–and–Bound (BB) method to implicitly enumerate all possible allocations while effectively pruning unnecessary paths in the search tree.

Module Scheduling (MS) algorithm to schedule the modules assigned to each PN subject to precedence constraints and latest module completion times.

The **BB** method enumerates all possible solutions to a given problem by "growing" the corresponding search tree. Each intermediate (leaf) vertex in the search tree corresponds to a partial (complete) allocation. This method is composed of two procedures: *branching* and *bounding*. The branching process generates the child vertices of an intermediate vertex x in the search tree, until an optimal solution is completely specified. Usually a dominance relation is derived to limit the number of child vertices generated for each intermediate vertex x without eliminating any path to an optimal solution. On the other hand, the bounding process calculates a tight upper bound of the objective function (UBOF) for each newly-generated vertex x, based on which one can decide whether or not x may lead to an optimal solution. If the UBOF of a vertex x is less than the current best objective found in the search process, then x will never lead to an optimal solution, and should thus be pruned. The interested reader is referred to [35] for a detailed account of the branch-and-bound method.

The **MA** scheme works as follows: All modules in the task system are assumed to be numbered in acyclic order, such that, if $M_i \to M_j$, then $i < j$. For example, the numbers which appear on the upper right corner of the boxes in Fig. 1b give an example of acyclic numbering. The **MA** scheme begins with a null allocation x_0 which corresponds to the root of the search tree, and allocates modules in the order of their acyclic numbering. Let $\mathbf{TG}(x)$ denote the set of modules which are already allocated under x,[5] and AN the set of active vertices in the search tree to be considered for expansion. (AN is determined by the *bounding test*.)

Expanding a vertex $x \in AN$ corresponds to allocating the module, M_i, with the smallest acyclic number in $\mathbf{TG} \setminus \mathbf{TG}(x)$ to a PN, where $\setminus$ denotes the difference of two sets. Only those PNs which have enough idle times to ensure the timely completion of M_i and survive the branching test will be considered as candidates for allocating M_i. After the expanding operation is performed, the bounding test is applied to those vertices expanded from x by allocating M_i to one of the candidate PNs. The UBOF, $\hat{P}_{ND}(y)$, of each newly-generated (intermediate) vertex y is calculated by scheduling modules $\in \mathbf{TG}(y)$ with **MS** and evaluating $P_{ND1}(y)$ and $P_{ND2}(y)$ with the expressions derived in Sections 4 and 5, respectively. If a vertex y has its $\hat{P}_{ND}(y)$ greater than the current best objective function value P^*_{ND}, it survives the bounding test, might possibly lead to the optimal solution, and will be made active and considered for vertex expansion in the next stage; otherwise, it will be pruned. The scheme terminates when an optimal solution is found.

The **MA** scheme is outlined below. The **MS** algorithm which schedules the modules assigned to each PN (under the *given* allocation and the given timing constraints) will be discussed in Section 4.1. The expressions of $P_{ND1}(y)$ and $P_{ND2}(y)$ (used to assess the *likelihood* of the timeliness and the logical correctness of an allocation y) will be derived in Sections 4.2 and 5, respectively. The branching and bounding tests used to achieve **BB** efficiency will be treated in Sections 6.1 and 6.2, respectively.

5. $\mathbf{TG}(x) = \mathbf{TG}$ if x is a complete allocation.

MA Algorithm:

Step 1. Generate the root, x_0, of the search tree, which corresponds to a null allocation. Set $AN := \{x_0\}$.

Step 2. Set $\mathbf{TG}(x_0) := \emptyset$, $x_{opt} := x_0$, and the objective function value achieved by x_{opt}, $P^*_{ND} = 0.0 = \hat{P}_{ND}(x_0)$.

Step 3. While $AN \neq \emptyset$ **do**

/* an optimal allocation has not yet been found */

Step 3.1. Node Selection Rule:

Step 3.1.1. Select the vertex $x \in AN$ with the largest $\hat{P}_{ND}(x)$.

Step 3.1.2. If $\hat{P}_{ND}(x) < P^*_{ND}$, terminate **MA**, and x_{opt} is the optimal solution. Otherwise, set M_i to be the module $\in \mathbf{TG} \setminus \mathbf{TG}(x)$ with the smallest acyclic number, and $AN := AN \setminus \{x\}$.

Step 3.2. Branching Test:

Step 3.2.1. Conduct the branching test on each PN. Only those PNs which survive the branching test will be considered for allocating M_i.

Step 3.2.2. Expand x by generating its valid child vertices, each of which corresponds to allocating M_i to one of the surviving PNs.

Step 3.3. Bounding Test: For each newly-generated vertex y,

Step 3.3.1. Use **MS** to find an optimal schedule for $\mathbf{TG}(y)$ under y and calculate the UBOF, $\hat{P}_{ND}(y)$.

Step 3.3.2. If $\hat{P}_{ND}(y) \le P^*_{ND}$, then prune y. Otherwise, the following two cases are considered:

Case 1. If y is a partial allocation, then set $AN := AN \cup \{y\}$, i.e., make y an active vertex.

Case 2. If y represents a complete assignment, $\hat{P}_{ND}(y)$ is the actual P_{ND} achieved under y. Since $\hat{P}_{ND}(y) > P^*_{ND}$, set $x_{opt} := y$ and $P^*_{ND} = \hat{P}_{ND}(y)$ to indicate that y has now become the best allocation found thus far.

4 Evaluation of Timeliness

In this section, we evaluate $P_{ND1}(x)$ for a given allocation x. We first describe how **MS** schedules all the modules assigned to a PN, say N_k, under x to minimize the maximum module tardiness subject to task release times and precedence constraints. By applying **MS** to each PN, we can obtain a module schedule under x. Second, we calculate the probability, $P(T_\ell$ is timely completed under $x)$, for every T_ℓ in **TG**. $P_{ND1}(x)$ can then be calculated from $P(T_\ell$ is timely completed under $x)$.

4.1 The Module Scheduling Algorithm

To facilitate the description and analysis of **MS**, we need to introduce the following notation:

- $\mathbf{TG}_c$: a component task graph of **TG**. If **TG** contains loops or OR-subgraphs, it will be replaced by a set of component task graphs without loops and OR-graphs before applying **MS** (see Section 4.2 for more on this). For the time-being, we only need to know that $\mathbf{TG}_c$ contains neither loops nor OR-subgraphs.

- $\mathbf{TG}_c(x)$: the set of modules $\in$ $\mathbf{TG}_c$ allocated under x.
- $S_k(x) = \{M_i : x_{ik} = 1\}$: the set of modules assigned to N_k under x.
- r_i: the release time of M_i, or the earliest time M_i can start its execution.
- LC_i: the latest completion time of M_i to ensure that <u>all</u> of its succeeding tasks will meet their deadlines.
- C_i: the completion time of M_i, which is determined by **MS**.
- e_i: the execution time of M_i.
- $\hat{e}_i$: the *modified* execution time of M_i, where

$$\hat{e}_i = \begin{cases} e_i & \text{if } M_i \text{ is scheduled to be executed upon its release at time } r_i \\ C_i - r_i & \text{otherwise.} \end{cases}$$

  $\hat{e}_i$ is used to include the effect of queuing M_i on the release times of all the modules that succeed M_i.
- $f_i(C_i)$: the cost incurred by completing M_i at C_i.
- $com_{ij}(x)$: the IMC time between M_i and M_j under x.
- d_{ij}: the IMC volume (measured in data units) between M_i and M_j.
- t_{mn}: the worst case link delay (measured in time units per data unit) of link ℓ_{mn}.
- $n(k, \ell)$: the number of edge-disjoint paths between N_k and N_ℓ.
- $I(m, n, k, \ell)$: the indicator variable such that $I(m, n, k, \ell) = 1$ if ℓ_{mn} lies on one of the $n(k, \ell)$ edge-disjoint paths between N_k and N_ℓ.
- $Y_{k\ell} = \frac{1}{n(k,\ell)} \sum_{m=1}^{K} \sum_{n=1}^{K} I(m,n,k,\ell) \cdot t_{mn}$: the worst case delay (measured in time units per data unit) between N_k and N_ℓ.
- B: the minimal set of modules that are processed without any idle time in $[r(B), c(B))$, where

$$r(B) = \min_{M_i \in B} r_i,\ c(B) = r(B) + e(B),$$

  and

$$e(B) = \sum_{M_i \in B} e_i.$$

- dg_i: the outdegree of M_i within a block of modules under consideration.

Specifically, $|S_k(x)|$ modules (possibly belonging to different tasks) are to be scheduled preemptively on N_k. Each module M_i becomes available upon its release at time r_i, which is initially set to the invocation time of the task to which M_i belongs. Precedence relations ($\rightarrow$) are considered in the entire task system: If $M_j \rightarrow M_i$, then M_i cannot start its execution before the completion of M_j, regardless of whether M_i and M_j are assigned to the same PN or not. Execution of a module may be preempted and then resumed later. Associated with each M_i is a monotone nondecreasing cost function $f_i(C_i)$. We want to find a schedule for the modules in $S_k(x)$ such that

$$f_{max}(S_k(x)) \overset{\Delta}{=} \max_{M_i \in S_k(x)} f_i(C_i)$$

is minimized. The schedule with the *minimal* cost $f^*_{max}(S_k(x))$ is said to be an *optimal* schedule of $S_k(x)$.

Before proceeding to describe and analyze **MS**, we define the cost function $f_i(C_i)$ and discuss how to calculate the two parameters, LC_i and r_i, $\forall i$. The cost function is defined as

$$f_i(C_i) = C_i - LC_i, \tag{4.1}$$

where LC_i is the latest time M_i must be completed to ensure the timeliness of all of its succeeding modules, and C_i is the completion time of M_i determined by **MS**. If $C_i > LC_i$, a positive cost will occur. Thus, with the definition of this cost function, minimizing the maximum cost function is equivalent to minimizing the maximum tardiness of modules in $\mathbf{TG}_c$.

The latest completion time, LC_i, of $M_i \in \mathbf{TG}_c$ is obtained as follows: Let LC_i be initially set to the deadline of the task to which M_i belongs. Then, modify LC_i as

$$LC_i = \min\left\{LC_i \min_j\left\{LC_j - e_j - com_{ij}(x) : M_i \rightarrow M_j\right\}\right\}, \quad i = N-1, \ldots, 1, \tag{4.2}$$

where the modules are assumed to be numbered in acyclic order and

$$com_{ij}(x) = \begin{cases} 0, & \text{if } M_i \text{ and } M_j \text{ are assigned to the same PN under } x, \\ d_{ij} Y_{k\ell}, & \text{if } M_i \text{ and } M_j \text{ are assigned to } N_k \text{ and } N_\ell \text{ under } x, \text{ respectively.} \end{cases}$$

Note that (4.2) computes backward from $i = N - 1$ to $i = 1$, because M_N has no successor by the nature of acyclic order, and, thus, the latest completion time of M_N is exactly the deadline of the task it belongs to. When x is a partial allocation and either M_i or M_j or both have not yet been assigned, $com_{ij}(x)$ is (optimistically) assumed to be 0.

The release time, r_i, of $M_i \in \mathbf{TG}_c(x)$ is obtained as follows: Let r_i be initially set to the invocation time of the task to which M_i belongs. Then, modify r_i as

$$r_i = \max\left\{r_i, \max_j\left\{r_j + \hat{e}_j + com_{ji}(x) : M_j \rightarrow M_i\right\}\right\}, \quad 2 \le i \le N, \tag{4.3}$$

where r_1 is the invocation time of the task to which M_1 belongs, and $\hat{e}_j = \max\{C_j - r_j, e_j\}$ is the *modified* execution time which equals the sum of M_j's execution time, e_j, and M_j's queuing time (if M_j is not scheduled to be executed upon its release). $\hat{e}_j$ is used to include the effect of queuing M_i's preceding module, M_j, on M_i's release time.

Note that the modified execution times of all M_i's preceding modules must be available prior to the calculation of r_i. This is ensured by allocating the modules in the order of their acyclic numbers. When an intermediate vertex y survives the bounding test and is put in AN, all modules in $\mathbf{TG}_c(y)$ would have been scheduled and their completion times (and, thus, modified execution times) would have been determined in the bounding process in the previous stage (Step 3.3 in **MA** in Section 3). Thus, when x is expanded from its parent vertex y by adding the new assignment of M_i, the schedules, completion times and modified execution times of all modules in $\mathbf{TG}_c(y)$ (which includes all preceding modules of M_i) must have been determined. So, all the e_j's needed in (4.3) are known at the time of calculating r_i.

EXAMPLE 1. Fig. 2 shows an example of how r_is and LC_is are calculated. The allocation x in Fig. 2 assigns M_1, M_3, M_5, and M_6 to N_1, and the other modules to N_2. Both module execution times and task deadlines are specified in the figure. The worst case IPC delay is assumed to be 0.5 unit of time, i.e., $d_{ij}Y_{k\ell} = 0.5$ wherever applicable. For example, the release time r_4 of M_4 is calculated as

$$r_4 = \max\left\{r_4, r_1 + \hat{e}_1 + com_{14}(x), r_2 + \hat{e}_2 + com_{24}(x), r_3 + \hat{e}_3 + com_{34}(x)\right\}$$
$$= \max\left\{0,\ 0 + 1 + 0.5,\ 0 + 1,\ 1.5 + 2 + 0.5\right\} = 4,$$

and the latest completion time, LC_{12}, of M_{12} is calculated as

$$LC_{12} = \min\left\{LC_{12}, LC_{13} - e_{13} - com_{12\ 13}(x)\right\}$$
$$= \min\left\{10,\ 12,\ -1\right\} = 10.$$

Now, we describe the **MS** algorithm, the theoretical base of which is grounded on the result of [36]. First, we arrange

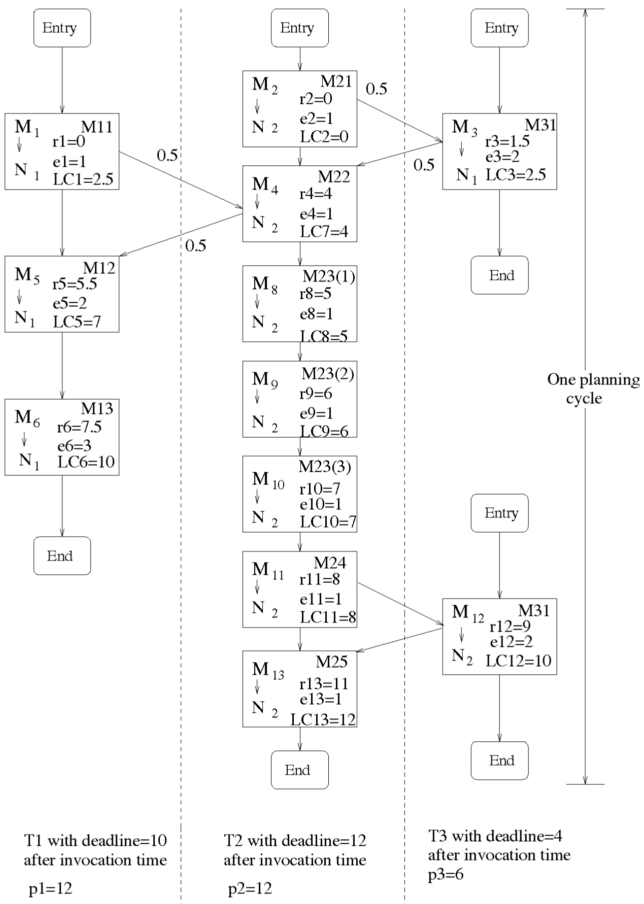


Fig. 2 . An example showing how r_i s and LC_i s are computed. All tasks are first invoked at time 0. (In this particular example, $\hat{e}_j = e_j$, $1 \le j \le 12$.)

the modules $\in S_k(x)$ in the order of nondecreasing release times. We then decompose $S_k(x)$ into blocks, where a block $B \subset S_k(x)$ is defined as the minimal set of modules processed without any idle time from $r(B) = \min_{M_i \in B} r_i$ until $c(B) = r(B) + e(B)$, where $e(B) = \sum_{M_i \in B} e_i$. That is, each $M_i \notin B$ is either completed no later than $r(B)$ or not released before $c(B)$. For example, as shown in Fig. 3, the set of modules assigned to N_1 in Fig. 2, $S_1(x) = \{M_1, M_3, M_5, M_6\}$, can be decomposed into three blocks, while the set of modules assigned to N_2, $S_2(x) = \{M_2, M_4, M_8, M_9, M_{10}, M_{11}, M_{12}, M_{13}\}$, can be decomposed into two blocks.

Obviously, scheduling modules in a block B is irrelevant to that in other blocks, so we can consider each block separately. Let dg_i denote the *outdegree* of M_i within B, i.e., the number of modules $M_j \in B$ such that $M_i \to M_j$. For each block B, we first determine the set $\hat{B} \overset{\Delta}{=} \{M_i : M_i \in B, dg_i = 0\}$, i.e., modules without successors in B, and then select a module M_m such that

$$f_m\big(c(B)\big) = \min_{M_i \in \hat{B}} f_i\big(c(B)\big), \tag{4.4}$$

i.e., M_m has no successor within B and incurs a minimum cost if it is completed last in $\hat{B}$. (In case of a tie, we choose the module with the largest acyclic number.) Now, consider an optimal schedule for the modules in B subject to the restriction that M_m is processed only if no other module is waiting to be processed. This optimal schedule consists of two parts:

Sched1: An optimal schedule with the cost $f^*_{max}(B - \{M_m\})$ for the set $B - \{M_m\}$, which could be decomposed into a number of subblocks $\hat{B}_1, \hat{B}_2, \ldots, \hat{B}_{\hat{b}}$.

Sched2: A schedule for M_m, which is given by

$$\big[r(B), c(B)\big] - \bigcup_{j=1}^{\hat{b}} \Big[r\big(\hat{B}_j\big), c\big(\hat{B}_j\big)\Big],$$

where $r(B) = \min_{M_i \in B} r_i$ and $c(B) = r(B) + e(B)$ with $e(B) = \sum_{M_i \in B} e_i$.

For this optimal schedule, we have

$$f^*_{max}(B) \text{ with the above restriction } = \max\Big\{f_m\big(c(B)\big), f^*_{max}\big(B - \{M_m\}\big)\Big\} \le f^*_{max}(B), \tag{4.5}$$

where the last inequality comes from:

1) $f^*_{max}(B) \overset{\Delta}{=} \min \max_{M_i \in B} f_i(C_i) \ge \min_{M_i \in B} f_i(c(B)) = \min_{M_i \in \hat{B}} f_i(c(B)) = f_m(c(B))$ by the way $\hat{B}$ was constructed from B and (4.4).
2) Since $B - \{M_i\}$ is a subset of B, $f^*_{max}(B) \ge f^*_{max}(B - \{M_i\})$, $\forall M_i$.

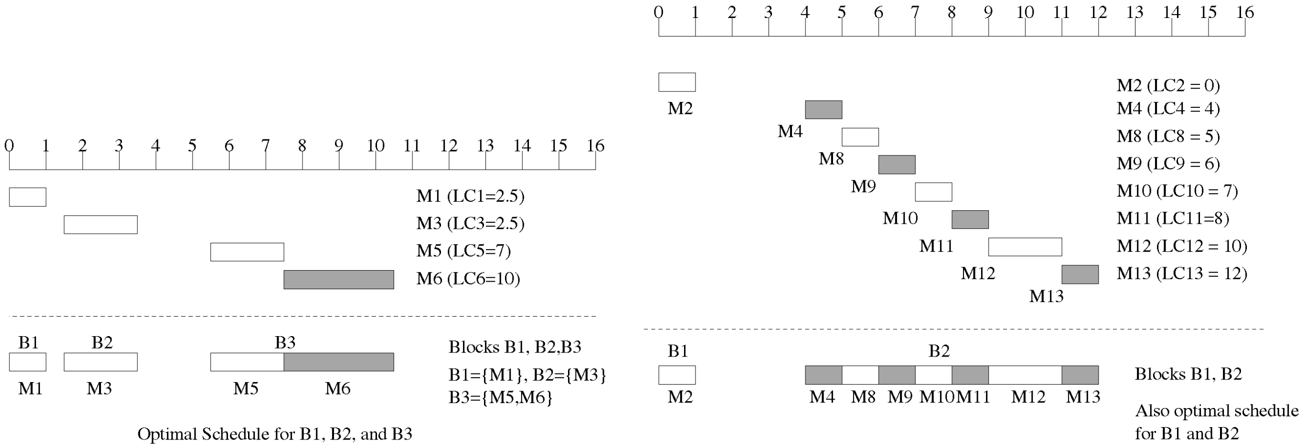


Fig. 3. Optimal schedule on N_1 under allocation x in which M_1, M_3, M_5, and M_6 are assigned to N_1, while the other modules are assigned to N_2.

It follows from (4.5) that there exists an optimal schedule in which M_m is scheduled only if no other module is waiting to be scheduled. By repeatedly and recursively applying the above procedure to each of the subblocks $\hat{B}_1, \hat{B}_2, \ldots, \hat{B}_b$, we obtain an optimal schedule for B. The rationale behind **MS** is that a PN is never left idle when there are modules ready to execute, and, by virtue of the cost function defined, it is always the module M_i with the smallest LC_i that will be executed among all released modules.

EXAMPLE 2. Take the task graph in Fig. 2 as an example, and consider the schedule on N_1. As shown in Fig. 3, $S_1(x)$ is composed of three blocks, $B_1 = \{M_1\}$, $B_2 = \{M_3\}$, and $B_3 = \{M_5, M_6\}$. The optimal schedule can be readily obtained (Fig. 3), since the schedules for B_1 and B_2 are trivial, and, for B_3, we have $\hat{B}_3 = \{M_6\}$, meaning that M_6 has to be processed only when no other module is waiting to be processed.

EXAMPLE 3. Fig. 4 gives another (more complicated) illustrative example, showing how **MS** schedules the modules assigned to a PN. r_i and LC_i, $1 \le i \le 5$, are assumed to have been computed from the entire task graph and are given in the figure. By ordering the modules according to their increasing release times, we obtain two blocks: $B_1 = \{M_1, M_2, M_3, M_4\}$ from [0, 8] (i.e., $r(B_1) = 0$, $e(B_1) = 8$, and $c(B_1) = 8$) and $B_2 = \{M_5\}$ from [9, 11] (i.e., $r(B_2) = 9$, $e(B_2) = 2$, and $c(B_2) = 11$). The schedule for B_2 is trivial, because B_2 consists of a single module and itself represents an optimal schedule for B_2. For B_1 we have $\hat{B}_1 = \{M_3, M_4\}$ and select M_3 to be processed only when no other modules are waiting, since $LC_3 > LC_4$. Now, $B - \{M_3\}$ consists of two subblocks: $B_{11} = \{M_1, M_2\}$ from [0, 3] and $B_{12} = \{M_4\}$ from [4, 6]. B_{12} itself represents an optimal schedule. For B_{11}, we have $\hat{B}_{11} = \{M_1, M_2\}$ and select M_1 to be processed last, since $LC_1 > LC_2$. The final optimal schedule for B_1 is obtained by combining the optimal schedule for B_{11} and B_{12} (**Sched1**) and the schedule for M_3 (**Sched2**) which consists of $[0, 8] - [0, 3] \cup [4, 6]$. The result is depicted in the last row of Fig. 4.

The **MS** algorithm along with the time complexity in each step is summarized below.

MS Algorithm:

Step 1: Compute the latest completion time LC_i, $1 \le i \le N$, for $\mathbf{TG}_c$. This computation requires $O(N^2)$ time.

Step 2: Compute the release time r_i for $M_i \in \mathbf{TG}_c(x)$ with respect to their precedence constraints. This computation, in the worst case, requires $O(N^2)$ time.

Step 3: Construct the blocks $B_1, B_2, \ldots, B_b$ of $S_k(x)$ for every N_k by ordering the modules $\in S_k(x)$ according to their nondecreasing release times. This ordering requires $O(|S_k(x)| \cdot log\ |S_k(x)|)$ time, $\forall k$.

Step 4: For each block B_i, $1 \le i \le b$, update the outdegree, dg_j, of every $M_j \in B_i$. This update requires $O(|S_k(x)|^2)$ time for all B_is $\subset S_k(x)$.

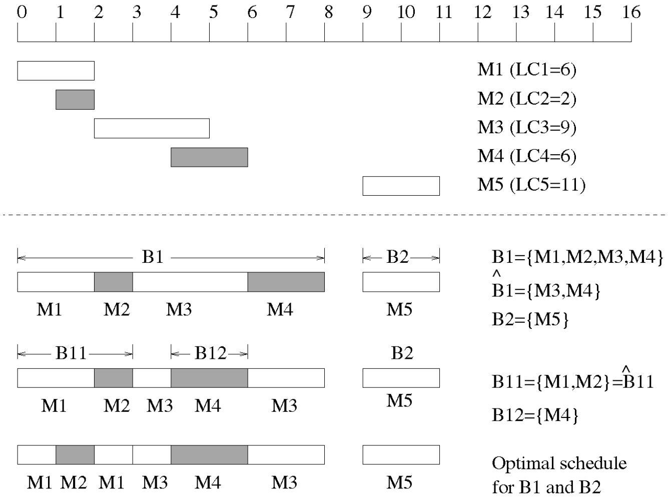


Fig. 4. An example showing how **MS** schedules the modules assigned to a PN. Note that all modules make their latest completion times under the optimal schedule.

Step 5: For each block B_i, select $M_m \in B_i$ subject to (4.4), determine the subblocks of $B_i - \{M_m\}$, and construct the schedule for M_m as given in Sched 2. Then, update the dg_j of every $M_j \in B_i - \{M_m\}$ with respect to the subblock of $B_i - \{M_m\}$ to which M_j belongs. By repeatedly applying Step 5 to each of the subblocks of $B_i - \{M_m\}$, one can obtain an optimal schedule. The time complexity for all repeated applications of Step 5 is bounded by $O(|S_k(x)|^3)$.

Since the time complexity associated with each step is polynomial, the **MS** algorithm is a polynomial algorithm.

4.2 Calculation of $P_{ND1}(x)$

We are now in a position to discuss how to calculate $P(T_\ell$ is timely completed under $x)$. Conceptually, given **TG** and x, we can determine the set, $S_k(x)$, of modules $\in$ **TG** assigned to N_k and, then, use **MS** to schedule modules in $S_k(x)$, $\forall k$. The completion time(s) of the last module(s) in $T_\ell \cap \mathbf{TG}$ under these schedules determines whether T_ℓ can be completed in time or not. However, since **TG** may contain loops and/or OR-subgraphs, the release times and the latest completion times of modules needed in Step 3 of **MS** may not be readily determined. Moreover, one cannot determine which module of T_ℓ to execute last if the last component in T_ℓ is an OR-subgraph.

4.2.1 Component Graphs

To resolve the above problem, we must eliminate the loops/OR-subgraphs in **TG** while retaining all the timing and probabilistic properties of **TG**. We first calculate the latest completion time, LC_i, of $M_i \in$ **TG** using (4.2), assuming that

A1. Every OR-subgraph following M_i, if any, is viewed as an AND-subgraph by ignoring branching probabilities.

A2. Every loop L_a following M_i, if any, is replaced by a cascade of n_{L_a} copies of its loop body, where n_{L_a} is the maximum loop count.

With **A1** and **A2**, the LC_is calculated are the worst-case latest completion time.

Second, we represent each loop $L_a \in$ **TG** with the cascaded m copies of its loop body with probability $(1-q_a)q_a^{m-1}$, where $1 \le m \le n_{L_a}$, and q_a is the looping-back probability of L_a. The last copy of $M_i \in L_a$ bears the LC_i calculated above, while the $(n_{L_a} - j)$th copy of M_i bears the latest completion time $LC_i - j \cdot e(L_a)$, where $e(L_a)$ is the execution time of the loop body. Also, we represent each OR-subgraph $O_b \in$ **TG** with its nth branch with probability $q_{b,n}$, where $1 \le n \le n_{O_b}$, $q_{b,n}$ is the branching probability of the nth branch of O_b, and n_{O_b} is the number of branches in O_b.

The **TG** can then be represented by the set of all possible combinations—which is termed as the set of *component task graphs*. For example, if there exists a loop L_a and an OR-subgraph O_b in **TG**, then there are a total of $n_{L_a} \times n_{O_b}$ component graphs of **TG**, and, with probability $p_c = (1-q_a)q_a^{m-1} \cdot q_{b,n}$, the **TG** is represented by the **TG**, with L_a replaced by the cascaded m copies of its loop body and O_b replaced by its nth branch. (One can trivially extend this to the case where there is more than one loop and/or OR-subgraph.)

For each component graph, $\mathbf{TG}_c$, of **TG**, we then calculate the release time, r_i, of $M_i \in \mathbf{TG}_c$ using (4.3). Using the r_is and LC_is determined above, we can apply Steps 3-5 in **MS** to find the best schedules for all modules in $\mathbf{TG}_c$. Note that, in a component graph $\mathbf{TG}_c$, the release time, r_i, and the number of times M_i is executed are both fixed, making it possible to decompose $S_k(x)$ into blocks.

For real-time applications in which the worst-case performance is the main concern or, for a **TG** which contains a large number of loops and/or OR-subgraphs, we can represent the **TG** with the set of single component task graphs in which each loop L_a is replaced by the cascaded n_{L_a} copies of the loop body (while each OR-subgraph is replaced by one of its branches). This significantly reduces the number of component graphs needed to be considered.

4.2.2 Calculation of $P_{ND}(x)$

We now calculate, for every T_ℓ in a component task graph $\mathbf{TG}_c$, the probability $P(T_\ell$ is timely completed under $x)$. Let the critical time of $M_i \in T_\ell$, D_i, be defined as the latest time M_i should be completed for the timely completion of the task T_ℓ only. Note that D_i can be obtained in the same way as LC_i, except that the precedence relations, $M_i \to M_j$ when $M_j \notin T_\ell$, are ignored. That is, let D_i be initially set to the deadline of T_ℓ to which M_i belongs. Then, D_i is modified as:

$$D_i = \min\left\{D_i, \min_{M_j \in T_\ell}\left\{D_j - e_j - com_{ij}(x) : M_i \to M_j\right\}\right\},$$
$$i = N-1, N-2, \ldots, 1.$$

Obviously, $D_i \ge LC_i$. Also, let

$$\hat{T}_\ell \overset{\Delta}{=} \left\{M_i : M_i \in T_\ell \cap \mathbf{TG}_c, dg_i = 0 \text{ with respect to } T_\ell \cap \mathbf{TG}_c\right\}$$

be the set of modules without any successor in $T_\ell \cap \mathbf{TG}_c$. Then, T_ℓ can be timely completed under the allocation x in the component task graph $\mathbf{TG}_c$ if $D_i \ge C_i$ for every module M_i which has no successor in $T_\ell \cap \mathbf{TG}_c$ (i.e., $\forall M_i \in \hat{T}_\ell$). In other words, the probability $P(T_\ell$ is timely completed under x in $\mathbf{TG}_c)$ can be expressed as

$$P\left(T_\ell \text{ is timely completed under } x \text{ in } \mathbf{TG}_c\right) = \prod_{M_i \in \hat{T}_\ell} \delta\left(D_i - C_i\right), \quad (4.6)$$

where $\delta(\cdot)$ is the step function, i.e., $\delta(t) = 1$ for $t \ge 0$, and $\delta(t) = 0$, otherwise. Consequently, $P(T_\ell$ is timely completed under $x)$ can be expressed as

$$P\left(T_\ell \text{ is timely completed under } x\right) = \sum_{\text{all } \mathbf{TG}_c'\text{s}} p_c \cdot P\left(T_\ell \text{ is timely completed under } x \text{ in } \mathbf{TG}_c\right), \quad (4.7)$$

where p_c is the probability that **TG** is represented by $\mathbf{TG}_c$. Finally, P_{ND1} can be expressed as

$$P_{ND2}(x) = \prod_{\ell=1}^{N_T} P\left(T_\ell \text{ is timely completed under } x\right), \quad (4.8)$$

where N_T, as defined in Section 2, is the number of periodic tasks in the system.

5 Evaluation of Logical Correctness

In this section, we calculate the probability, $P_{ND2}(x)$, that:

1) All PNs are operational during the execution of modules assigned to them, and
2) All communication links are operational during the course of IPCs.

he derivation of $P_{ND2}(x)$ is similar to that of the reliability function in [18], [19], but we relax the following two unrealistic assumptions used in [18], [19]:

A1) The network topology is cycle-free, i.e., there is one and only one path between any pair of PNs;
A2) Each module is executed only once in a task invocation.

Instead of the first assumption, we allow an arbitrary network topology, and also allow the IPCs between N_k and N_ℓ to take place over one of the (arbitrarily chosen) edge-disjoint paths between the two PNs. In contrast to the second assumption, we allow modules to be contained in loops and / or branches of OR-subgraphs, i.e., modules may be executed more than once, or not executed at all in a task invocation.

To facilitate the derivation of $P_{ND2}(x)$, we need the following notation:

- LP: the set of modules which are contained in loops.
- OR: the set of modules which are on the branches of OR-subgraphs.
- q_a: the looping-back probability of loop L_a.
- $q_{b\ell}$,: the branching probability of the ℓth branch of an OR-subgraph, O_b.
- n_{L_a} : the maximum count of loop L_a.
- n_{O_b} : the number of branches in an OR-subgraph O_b.
- λ_k: the constant exponential failure rate of N_k.
- $\hat{\lambda}_{mn}$: the constant exponential failure rate of link ℓ_{mn}. Failure occurrences are assumed to be statistically independent of one another.
- $R_{mn}(i, j, n_c, x)$: the probability that link ℓ_{mn} is operational during the n_c occurrences of IMC between M_i and M_j under allocation x.
- $R_{pn}(x)$: the probability that all PNs are operational during the execution of modules assigned to them under x.
- $R_{link}(x)$: the probability that all links are operational for all IMCs under x.

Under allocation x, the probability that all PNs remain fault-free during the execution of the modules assigned to them is:

$$\begin{aligned} R_{pn}(x) = \{ &\prod_{M_i \notin LP \cup OR} \prod_{k=1}^{K} \exp\left(-\lambda_k x_{ik} e_i\right)\} \cdot \\ \{ &\prod_{L_a \in LP} \sum_{\ell=1}^{n_{L_a}} \left[\left(1-q_a\right) q_a^{\ell-1} \cdot \prod_{M_i \in L_a} \prod_{k=1}^{K} \exp\left(-\lambda_k x_{ik} \ell e_i\right) \right] \} \cdot \\ \{ &\prod_{O_b \in OR} \sum_{\ell=1}^{n_{O_b}} \left[q_{b,\ell} \cdot \prod_{\substack{M_i \in \ell\text{th} \\ \text{branch of } O_b}} \prod_{k=1}^{K} \exp\left(-\lambda_k x_{ik} e_i\right) \right] \}. \end{aligned} \quad (5.1)$$

Note that all factors, except the one associated with $x_{ik} = 1$ in the term $\prod_{k=1}^{K} \exp(-\lambda_k x_{ik} e_i)$, reduce to one. The expression within the first pair of braces is the probability that the PNs on which stand-alone modules[6] reside are operational during the execution of these modules. Similarly, the expression in the second (third) pairs of braces is the probability that the PNs on which the modules in loops (OR-subgraphs) reside are operational during the execution of these modules. In case M_i is contained in a loop L_a, with probability $q_a^{\ell-1}(1-q_a)$, M_i requires an execution time $\ell \cdot e_i$, and, in case M_i is on the ℓth branch of an OR-subgraph, O_b, with probability $q_{b,\ell}$, M_i will be executed. Note that (5.1) can be readily extended to the case where a loop / OR–subgraph is contained in other loops and / or OR-subgraphs.

Example 4. Consider Fig. 1b as an example, where $LP = \{M_8\}$ and $OR = \{M_6, M_7\}$. If all modules of T_1 are assigned to N_1 and all modules of T_2 and T_3 are assigned to N_2, i.e., $x_{11} = x_{51} = x_{61} = x_{71} = 1$ and $x_{22} = x_{32} = x_{42} = x_{82} = x_{92} = x_{102} = x_{112} = 1$, then

$$R_{pn}(y) = e^{-\lambda_1(e_1+e_5)} \cdot e^{-\lambda_2(e_2+e_3+e_4+e_9+e_{10}+e_{11})} \cdot$$

$$\left\{qe^{-\lambda_1 e_0} + (1-q)e^{-\lambda_1 e_7}\right\} \cdot \sum_{\ell=1}^{n_L} (1-p)p^{\ell-1} e^{-\lambda_2 \ell e_8}.$$

The expression of $R_{link}(x)$ calls for the derivation of the probability that link ℓ_{mn} is operational during the n_c occurrences of IMC between M_i and M_j under x, $R_{mn}(i, j, n_c, x)$:

$$R_{mn}\left(i, j, n_c, x\right) = \prod_{k=1}^{K} \prod_{\ell=1, \ell\neq k}^{K} \exp\left(-\hat{\lambda}_{mn} \cdot \frac{n_c t_{mn} d_{ij}}{n(k,\ell)} \cdot x_{ik} x_{j\ell} \cdot I(m,n,k,\ell)\right). \quad (5.2)$$

Two remarks are in order:

- All the $K(K-1)$ terms in (5.2)—except for the term corresponding to $x_{ik} = x_{j\ell} = 1$—reduce to one.
- If M_i is assigned to N_k ($x_{ik} = 1$), M_j is assigned to N_ℓ ($x_{j\ell} = 1$), and ℓ_{mn} lies on one of the edge-disjoint paths between N_k and N_ℓ ($I(m, n, k, \ell) = 1$), then

$$R_{m,n}\left(i, j, n_c, x\right) = \exp\left(-\hat{\lambda}_{mn} \frac{n_c t_{mn} d_{ij}}{n(k,\ell)}\right),$$

where $\frac{n_c t_{mn} d_{ij}}{n(k,\ell)}$ is the (average) worst case communication

6. Modules which are contained in neither loops nor OR-subgraphs in the TG.

time over link ℓ_{mn} contributed by the n_c occurrences of IMC between M_i and M_j.

EXAMPLE 5. Given the simple distributed system represented by a complete graph of three PNs (where $n(k, \ell) = 2$ and $I(m, n, k, \ell) = 1$, $1 \le m, n \le 3$, $m \ne n$, $1 \le k, \ell \le 3$, $k \ne \ell$) and the TG in Fig. 1b, we have

$$R_{12}(i, j, n_c, x) = \exp\left(-\hat{\lambda}_{12} \cdot \frac{n_c t_{12} d_{ij}}{2} \cdot \left(x_{i1}x_{j2} + x_{i1}x_{j3} + x_{i2}x_{j1} + x_{i2}s_{j3} + x_{i3}x_{j1} + x_{i3}x_{j2}\right)\right).$$

Under the allocation x in which modules of T_1 and $T_2 \cup T_3$ are assigned to N_1 and N_2, respectively,

$$R_{12}(1,\ 4, n_c, x) = \exp\left(-\hat{\lambda}_{12} \cdot \frac{n_c t_{12} d_{14}}{2}\right);$$

since $x_{11}x_{42} = 1$, and $R_{12}(2, 4, n_c, x) = 1$, i.e., the IMCs between M_2 and M_4 are accomplished via shared memory and do not use link ℓ_{12} at all.

Now, we are in a position to derive the expression of $R_{link}(x)$. Let

$$\mathbf{C}_1 \overset{\Delta}{=} \left\{(M_i, M_j) : d_{ij} > 0, \text{ and neither of } M_i \text{ and } M_j \text{ resides in a loop or an OR - subgraph}\right\},$$

$$\mathbf{C}_2 \overset{\Delta}{=} \left\{(M_i, M_j) : d_{ij} > 0, \text{ and one of } M_i \text{ or } M_j \text{ resides in a loop while the other (not contained in an OR - subgraph) resides immediately before or afterthe loop}\right\},$$

$$\mathbf{C}_3(L_a) \overset{\Delta}{=} \left\{(M_i, M_j) : d_{ij} > 0 \text{ and both } M_i \text{ and } M_j \text{ reside in the loop } L_a\right\}, \forall L_a \in LP;$$

$$\mathbf{C}_4(O_b, \ell) \overset{\Delta}{=} \left\{(M_i, M_j) : d_{ij} > 0 \text{ and either } M_i \text{ or } M_j \text{ or both reside on the } \ell \text{th branch of } O_b\right\}, \forall O_b \in OR,\ 1 \le \ell \le n_{O_b},$$

where $d_{ij} > 0$ indicates that M_i and M_j communicate with each other. The rationale behind defining $\mathbf{C}_1$, $\mathbf{C}_2$, $\mathbf{C}_3(L_a)$, and $\mathbf{C}_4(O_b, \ell)$ is as follows: If $(M_i, M_j) \in \mathbf{C}_1 \cup \mathbf{C}_2$, then M_i and M_j communicate with each other exactly once. If $(M_i, M_j) \in \mathbf{C}_3(L_a)$, then M_i and M_j communicate n_c times (each with data amount d_{ij}) with probability $q_a^{n_c-1}(1-q_a)$, where q_a is the looping-back probability of loop L_a. On the other hand, if $(M_i, M_j) \in \mathbf{C}_4(O_b, \ell)$, then M_i and M_j communicate once with probability $q_{b,\ell}$, where $q_{b,\ell}$ is the branching probability of the ℓth branch of the OR-subgraph O_b. $R_{link}(x)$ can then be expressed as:

$$R_{link}(x) = \{\prod_{(M_i,M_j)\in \mathbf{C}_1 \cup \mathbf{C}_2} \prod_{\ell_{mn}} R_{mn}(i, j,\ 1, x)\} \cdot \{\prod_{L_a \in LP} \sum_{n_c=1}^{n_{L_a}} q_a^{n_c-1}(1-q_a) \cdot [\prod_{(M_i,M_j)\in \mathbf{C}_3(L_a)} \prod_{\ell_{mn}} R_{mn}(i, j, n_c, x)]\} \cdot \{\prod_{O_b \in OR} \sum_{\ell=1}^{n_{O_b}} q_{b,\ell} \cdot [\prod_{(M_i,M_j)\in \mathbf{C}_4(O_b,\ell)} \prod_{\ell_{mn}} R_{mn}(i, j,\ 1, x)]\}. \tag{5.3}$$

For clarity of presentation, (5.3) excludes the case where a loop / OR-subgraph is contained in other loops and / or OR-subgraphs. However, it is straightforward to extend (5.3) to include such a case.

EXAMPLE 6. Consider again the example of allocating the TG in Fig. 1b to the distributed system represented by a three-complete graph. We have $\mathbf{C}_1 = \{(M_1, M_4), (M_1, M_5), (M_2, M_3), (M_2, M_4), (M_3, M_4), (M_4, M_5), (M_9, M_{10}), (M_9, M_{11}), (M_{10}, M_{11})\}$, $\mathbf{C}_2 = \{(M_4, M_8), (M_8, M_9)\}$, $\mathbf{C}_3 = \emptyset$, $\mathbf{C}_4(O_1, 1) = \{(M_1, M_6)\}$, and $\mathbf{C}_4(O_1, 2) = \{(M_1, M_7)\}$. Thus,

$$R_{link}(x) = \{\prod_{(M_i,M_j)\in \mathbf{C}_1 \cup \mathbf{C}_2} \prod_{\ell_{mn}} R_{mn}(i, j,\ 1, x)\} \cdot \{q\prod_{\ell_{mn}} R_{mn}(1,\ 6,\ 1, x) + (1-q)\prod_{\ell_{mn}} R_{mn}(1,\ 7,\ 1, x)\}.$$

Under the allocation x given in Example 5, we have

$$R_{link} = \prod_{\ell_{mn}} R_{mn}(1,\ 4,\ 1, y) \cdot \prod_{\ell_{mn}} R_{mn}(4,\ 5,\ 1, y) = e^{-\left(\hat{\lambda}_{12}t_{12} + \hat{\lambda}_{13}t_{13} + \hat{\lambda}_{32}t_{32}\right)d_{14}/2} \cdot e^{-\left(\hat{\lambda}_{12}t_{12} + \hat{\lambda}_{13}t_{13} + \hat{\lambda}_{32}t_{32}\right)d_{45}/2},$$

where the first and the second factors are contributed by the IMC between M_1 and M_4 and between M_4 and M_5, respectively. For example, $e^{-\hat{\lambda}_{12}t_{12}d_{14}/2}$ is contributed by the IPC between M_1 and M_4 which runs through ℓ_{12}, and $e^{-(\hat{\lambda}_{32}t_{32} + \hat{\lambda}_{13}t_{13})d_{14}/2}$ is contributed by the IPC which routes through ℓ_{13} and ℓ_{32}.

Finally, we have

$$P_{ND2}(x) = R_{pn}(x) \cdot R_{link}(x). \tag{5.4}$$

6 BRANCHING AND BOUNDING TESTS

The branching test uses the dominance relation derived from the requirement of timely completion of tasks to limit the number of child vertices generated in the branching process. The bounding test derives an UBOF for each intermediate vertex with which one decides whether or not to prune an intermediate vertex in the bounding process. In this section, we discuss how we design the branching and bounding tests.

6.1 Branching Test

Recall that expanding an intermediate vertex x in the search tree corresponds to allocating the module with the smallest acyclic number that has not yet been allocated (i.e., a module in $\mathbf{TG} \setminus \mathbf{TG}(x)$). The branching test uses the following dominance relation. M_i can be invoked after all its precedence constraints are met and must be completed by its latest completion time, LC_i, to ensure that all its succeeding tasks meet their deadlines. Hence, if

1) the idle time of a PN, say N_k, during the interval $[r_i, LC_i]$ is smaller than e_i, and
2) the module, say M_j, scheduled to be executed[7] on N_k in $[r_i, LC_i]$ under a partial allocation x has tighter timing constraints than M_i (so no preemption on N_k is possible to ensure the completion of M_i before LC_i),

then allocating M_i to N_k is likely to miss M_i's latest completion time. Thus, N_k should not be a candidate PN for allocating M_i, i.e., N_k fails the branching test.

Branching Test:

Step 1. Calculate optimistic estimates, r_i^o and LC_i^o, of r_i and LC_i, assuming that

A1. Every pair of communicating modules that have not yet been assigned (i.e., $\in$ $\mathbf{TG}\setminus\mathbf{TG}(x)$) reside on the same PN.

A2. The OR-subgraph preceding or following M_i, if any, is replaced by the branch with the smallest flowtime.

A3. The loop preceding, containing, or following M_i, if any, is replaced by its loop body (i.e., the loop executes only once).

Step 2. Calculate an pessimistic estimate, LC_i^p, of LC_i, assuming that

A4. The IMCs in $\mathbf{TG}\setminus\mathbf{TG}(x)$ are executed on N_k and N_ℓ with the largest nominal inter-PN delay $Y_{k\ell}$.

A5. The OR-subgraph following M_i is replaced by the branch with the largest flowtime.

A6. The loop following M_i (if any) is replaced by the cascaded n_L copies of its loop body, where n_L is its maximum loop count.

Step 3. For each N_k, check whether the following two conditions are true or not:

C1. The idle time of N_k in $[r_i^o, LC_i^o]$ is less than e_i.

C2. $LC_j \le LC_i^p$, where LC_j is the latest completion time of the module, M_j, scheduled last in $[r_i^o, LC_i^o]$ on N_k under the partial allocation x.

If both conditions are true, then N_k fails the test and is not considered for allocating M_i.

A1-A3 ensure $r_i^o \le r_i$ and $LC_i^o \ge LC_i$, thus making the interval $[r_i^o, LC_i^o]$ larger than $[r_i, LC_i]$. **A4-A6** ensure $LC_i^p \le LC_i$, making M_i likely to preempt other modules on N_k. Consequently, the use of the optimistic interval, $[r_i^o, LC_i^o]$, and the pessimistic value, LC_i^p, ensure that the PNs which fail the branching test cannot indeed complete M_i in time.

6.2 Calculation of an UBOF for the Bounding Test

The bounding test calculates an UBOF for each intermediate vertex, and prunes (keeps) the intermediate vertex when the calculated UBOF $\le$ (>) the best objective function, P_{ND}^*, found thus far. The bounding test uses the following principles. A vertex y is generated from its parent vertex x by adding the assignment $M_i \to N_k$ to the partial allocation x for some N_k that survives the branching test. After including $M_i \to N_k$, N_k needs to reschedule the modules assigned to it under y (i.e., the modules $\in S_k(y)$) using **MS**. Because modules are assigned in acyclic order, all preceding modules of M_i in $S_k(y)$ have their latest completion times < LC_i, and their schedules will not be changed by the addition of M_i. On the other hand, if some nonpreceding module M_j does change its schedule as a result of $M_i \to N_k$, then the release time(s) of all M_js succeeding module(s) have to be changed accordingly. Consequently, the PNs ($\ne N_k$) on which these succeeding modules of M_j reside need to reconsider their module schedules.

EXAMPLE 7. Fig. 5 gives an example of how adding $M_i \to N_k$ to a partial allocation might affect the schedules on other PNs. In the partial allocation x prior to the assignment $M_6 \to N_1$, M_1 and M_2 are assigned to N_1, and M_3, M_4, and M_5 are assigned to N_2. The optimal schedules on N_1 and N_2 for x (obtained from the bounding process of the last stage) are shown in Fig. 5a. Now, assign M_6 to N_1 to get the child vertex, y, of x. As shown in Fig. 5b, N_1 needs to reschedule its assigned modules. (Note that the schedule for M_1, however, does not change, since M_1 is a preceding module of M_6.) Also, the schedule change on N_1—especially, the schedule change for M_2—alters the release times of M_2's succeeding modules, M_4 and M_5. So, the schedule on N_2, the PN on which M_4 and M_5 reside, needs to be changed as well (Fig. 5b).

Let $\hat{PN}$ denote the set of PNs which need to reconsider their module schedules as a result of $M_i \to N_k$. Then, a UBOF is calculated by the following steps.

Calculation of a UBOF:

Step 1. Represent the **TG** with the set of component graphs.

Step 2. Calculate $\hat{P}_{ND1}(y)$ as follows:

Step 2.1. In each component graph $\mathbf{TG}_c$:

Step 2.1.1. Reschedule the modules $\in S_m(y)$ for every $N_m \in \hat{PN}$ using **MS** and **A1** in the branching test.

Step 2.1.2. Use (4.6) to calculate $P(T_\ell$ is timely completed under y in $\mathbf{TG}_c)$ for every $T_\ell \in \mathbf{TG}_c$, where $\hat{T}_\ell$, in (4.6) is modified as **A7.**

$$\hat{T}_\ell \overset{\Delta}{=} \{M_i : M_i \in T_\ell \cap \mathbf{TG}_c(y), dg_i = 0 \text{ w.r.t. } T_\ell \cap \mathbf{TG}_c(y)\},$$

i.e., $\mathbf{TG}_c$ in (4.6) is replaced by the set of modules $\in \mathbf{TG}_c$ allocated under y.

Step 2.2. Calculate $P(T_\ell$ is timely completed under $y)$ for

7. By "last," we mean the module is executed only if no other modules are waiting for processing on N_k.

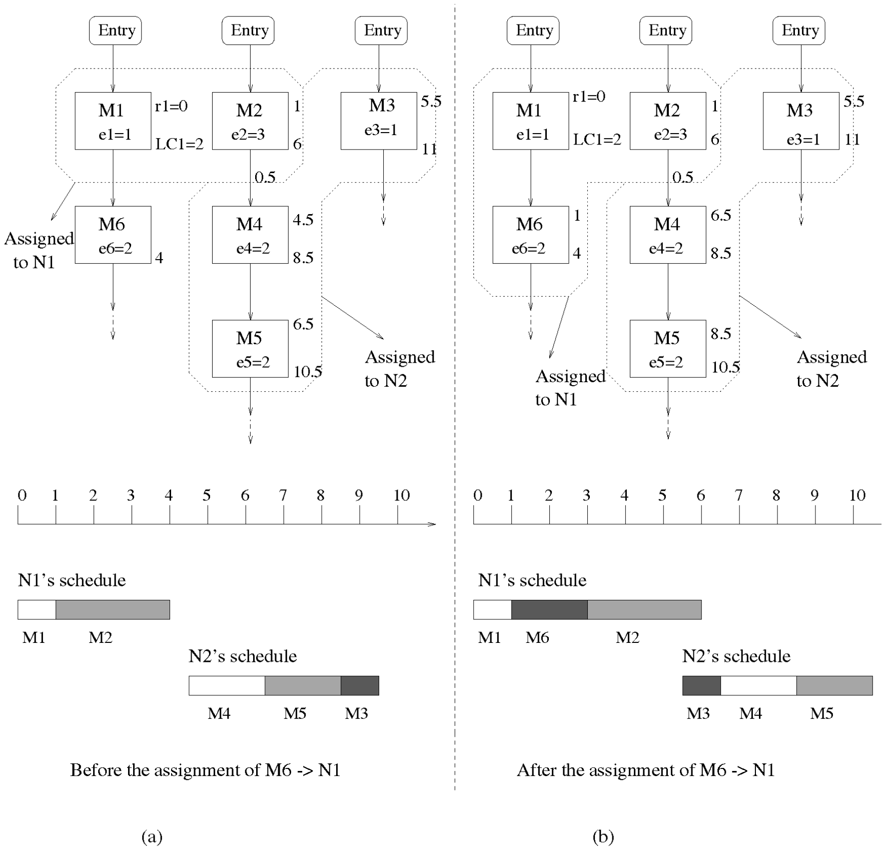


Fig. 5. An example showing how a new assignment $M_i \rightarrow N_k$ might affect the module schedule of $N_m \neq N_k$.

every $T_\ell \in$ **TG** and $\hat{P}_{ND1}(y)$, using (4.7) and (4.8), respectively.

Step 3. Calculate $\hat{P}_{ND2}(y)$ using (5.1), (5.3), and (5.4), and

A8. Every $M_j \in \mathbf{TG} \setminus \mathbf{TG}(y)$ is assumed to be allocated to the most reliable PN, and every pair of communicating modules in $\mathbf{TG} \setminus \mathbf{TG}(y)$ reside on the same PN.

Step 4. Calculate $\hat{P}_{ND}(y) = \hat{P}_{ND1}(y) \cdot \hat{P}_{ND2}(y)$.

Note that, because of the use of **A1** and **A7**, $\hat{P}_{ND1}(y)$ derived above is an upper bound of P_{ND1} of any leaf vertex (complete allocation) generated from y. Moreover, whether or not modules $\in \mathbf{TG} \setminus \mathbf{TG}(y)$ meet their latest completion times need not be considered in the calculation of $\hat{P}_{ND1}(y)$, and is thus excluded by **A7**.

7 Numerical Examples

The performance of **MA** is evaluated according to the following sequence:

1) discussion on the generation of task graphs and distributed systems;
2) the characteristics of **MA**;
3) the practicality of **MA**.

7.1 Generation of Task Graphs and Distributed Systems

There are a large number of parameters that may affect the performance of **MA**. They can be classified as *system parameters*, which specify the distributed system under consideration, and *task parameters*, which specify the TG. The generation of realistic TGs and distributed systems largely depends on how these parameters are specified. However, little is reported in the literature about "typical" real-time TGs and their communication patterns. Thus, we randomly generate both system and task parameters in our numerical experiments. We believe these randomly generated task graphs cover a wide spectrum of real-time applications.

The number of PNs in the distributed system is varied from three to 40. After the number of PNs is selected, the network topology is then arbitrarily generated. The worst case link delay, t_{mn}, associated with ℓ_{mn} is exponentially distributed with mean $0.1\bar{e}$, where $\bar{e}$ is the mean module execution time. The node failure rate, λ_i, and the link failure rate, $\hat{\lambda}_{mn}$, were varied from 10^{-5} to 0.5 (1/time unit). The number of modules, N, to be allocated is varied from four to 50. The execution time of a module is exponentially distributed with mean 1.0 unit of time. The IPC volume between two communicating modules is uniformly distributed over (0, 10] data units. The precedence constraints and the timing requirements of the TG are also randomly generated.

Before running experiments, we eliminated the TGs which were definitely infeasible. Infeasibility is detected by calculating release times and latest completion times of all modules, while ignoring all IMC times. If the interval between the latest completion time and the release time is less than the execution time for some module(s) in all the component graphs of a TG, this TG is infeasible (in the sense that some tasks cannot be completed in time even if infinite resources were available) and is not considered any further.

All experiments were performed on a Sun4 SPARC station running the SUNOS 4.1.3 operating system. Due to space limitation, we present only a few representative cases and statistical results. However, the conclusions drawn from the following summary were corroborated by all the experiments conducted.

7.2 Characteristics of MA

By virtue of the **BB** method, **MA** always yields the best allocation. To further examine the characteristics of the optimal allocation found by **MA**, experiments were performed on

1) TGs with different degrees of parallelism[8];
2) task sets with different degrees of deadline tightness;
3) distributed systems with different worst case link delays and node / link failure rates.

Several interesting properties observed in the experiments are given below.

P1. **MA** allocates sequentially-executing modules subject to the same tight timing constraints to the same PN. For example, the best allocation of the TG in Fig. 1b to the distributed system represented by a complete graph of three PNs and with homogeneous node failure rates (λ_k = 0.001) and link failure rates (λ_{mn} = 0.001) is to assign T_1 to N_1, and both T_2 and T_3 to N_2. **MA** recognizes that the execution path $M_2 \rightarrow M_3 \rightarrow M_4 \rightarrow M_8 \rightarrow M_9 \rightarrow M_{10} \rightarrow M_{11}$ in the TG is critical subject to T_3's deadline and cannot tolerate any IPC delay, thus allocating both T_2 and T_3 to the same PN. The resulting best objective function value is $P_{ND} = 9.8227 \times 10^{-1}$.

P2. Heavily communicating modules may not necessarily be allocated to the same PN. For example, consider the allocation of the TG in Fig. 6a. The attributes of both the TG and the distributed system are specified in Experiment I. As shown in Fig. 6b, **MA** allocates M_1, M_6, and M_7 to N_1; M_4, M_8, M_9, and M_{10} to N_2; M_2, M_3, and M_5 to N_3 so that all modules meet their latest completion times (P_{ND1} = 1.0) and are allocated to the most reliable PNs, N_1 – N_3 ($P_{ND2} = 9.7933 \times 10^{-1}$). Although the IMC between M_4 and M_5 is twice more than the others, M_4 and M_5 are allocated to different PNs. This is mainly because T_2 has a less tight timing constraint than others and can thus allow IPCs among its modules. This observation is in sharp contrast to the common notion that heavily communicating modules should always be co-allocated [23], [37].

P3. If the distributed system is homogeneous, **MA** assigns modules, subject to task timing constraints, in such a way that as few IPCs as possible will occur. To demonstrate this tendency, consider Experiment II in Fig. 6. The attributes of both the TG and the distributed system remain the same as in Experiment I, except that λ_4 = 0.001 (i.e., the distributed system becomes homogeneous). Now, the allocation and schedules specified in Fig. 6c give the best solution (P_{ND1} = 1.0, $P_{ND2} = 9.8096 \times 10^{-1}$). Note that the only IPC occurs between M_1 and M_3, which cannot be eliminated, because all modules of T_1 cannot be allocated to the same PN under T_1's timing constraint.

P4. If both timeliness and logical correctness cannot be achieved at the same time, **MA** maximizes P_{ND} by making a compromise between these two objectives. This is demonstrated by conducting three experiments: in Experiment I, the deadlines of the four tasks are set as d_1 = 5.0, d_2 = 6.0, d_3 = 6.0, and d_4 = 8.0. As shown in Fig. 6b, **MA** allocates modules only to (three) reliable PNs while meeting the timing constraints (P_{ND1} = 1.0, $P_{ND2} = 9.7933 \times 10^{-1}$). As the deadline constraints get tighter, in Experiment III, i.e., d_1 and d_2 remain unchanged while d_3 becomes 4.0 and d_4 becomes 7.5, **MA** is "forced" to allocate some of the modules (M_1) to a less reliable PN (N_4) in order to meet all timing constraints. Fig. 6c gives the best allocation and schedules: T_2 and T_3 are now allocated to N_1, T_4 to N_2, and T_1 to N_3 and N_4 (P_{ND1} = 1.0, $P_{ND2} = 9.6346 \times 10^{-1}$). On the other hand, if N_4 is highly prone to failure, as is assumed in Experiment IV (λ_4 is increased from 0.01 to 0.5), **MA** decides not to use N_4 at the risk of not making task T_4's deadline, as depicted in Fig. 6d (P_{ND1} = 0.7, $P_{ND2} = 9.8096 \times 10^{-1}$, and $P_{ND} = 6.8667 \times 10^{-1}$).

Note that, for ease of exposition, we choose the simple **TG** given above as the example task system. However, the observation made was corroborated by all the experiments conducted. See [38] for more experimental results.

7.3 Practicality of MA

To test the practicality of MA for reasonably large TGs and / or distributed systems, we ran experiments on

1) TGs with 4-50 modules, while varying module execution times, IMC volumes, task deadlines, and randomly generating precedence constraints;
2) distributed system topologies with 3-40 PNs, while randomly varying worst case link delays and the degree of network connectivity.

We then computed the ratio of the number of search-tree vertices visited to the total number, $K^{N+1} - 1$, of vertices in the search tree. The numerical results for different combinations of N and K are summarized in Table 1. The number of trials in each combination of N and K was determined,[9]

8. A TG is said to have a high-degree of parallelism if most of its modules can be executed in parallel when there are enough resources. This could occur if the TG contains AND-subgraphs with a large number of branches and / or most tasks in the TG do not communicate with one another so that only a few precedence constraints are imposed on the modules belonging to different tasks.

9. Under the assumption that the parameter to be estimated (i.e., the mean number of search–tree vertices visited) has a normal distribution with

TABLE 1
THE NUMBER AND PERCENTAGE OF VERTICES VISITED IN THE SEARCH TREE BY MA

N		6	8	10	15	20	30	40
Best	*# visited*	19	25	166	649	1488	2,432	3,576
case	*% visited*	0.87%	0.13%	0.09%	0.002%	—	—	—
Average	*# visited*	47	442	2,230	12,720	37,635	56,015	68,687
case	*% visited*	2.15%	2.25%	1.26%	0.03%	3.60×10^{-4}%	—	—
Worst	*# visited*	159	1240	14,680	79,905	150,304	268,420	374,572
case	*% visited*	7.27%	6.30%	8.29%	0.19%	1.44×10^{-3}%	—	—
$K^{N+1} - 1$		2,186	19,682	177,146	4.305×10^{7}	1.046×10^{10}	6.177×10^{14}	3.647×10^{19}

(a) $K = 3$

K		2	3	4	5	6	8	10
Best	*# visited*	21	166	233	426	581	705	814
case	*% visited*	1.03%	0.09%	5.56×10^{-3}%	8.72×10^{-4}%	1.60×10^{-4}%	—	—
Average	*# visited*	360	2,230	7,452	9,388	11,068	17,038	27,026
case	*% visited*	17.59%	1.26%	0.178%	0.019%	3.05×10^{-3}%	1.98×10^{-3}%	—
Worst	*# visited*	723	14,680	35,335	50,353	56,671	63,248	83,035
case	*% visited*	35.32%	8.29%	0.842%	0.103%	0.016%	7.36×10^{-4}%	—
$K^{N+1} - 1$		2047	177,146	4.194×10^{6}	4.883×10^{7}	3.628×10^{8}	8.590×10^{9}	1.00×10^{11}

(b) $N = 10$

— indicates less than $10^{-6} \times 100$% of nodes in the search-tree were visited.

so that a 95 percent (90 percent) confidence level may be obtained for a maximum error within 10 percent of the average numbers reported for $N \le 10$ ($N > 10$). Also given in each combination are the worst and best results ever found in these trials.

In all the experiments conducted, no more than 9 percent of the search-tree vertices were visited before finding the best allocation for $N \ge 6$ and $K \ge 3$. Also, the percentage of search-tree vertices visited falls drastically as N and/or K grows, as shown in Table 1. This is because the "increasing rate" for the number of vertices visited as N and/or K grows is far lower than exponential. This suggests that both the dominance relation and the UBOF derived effectively prune unnecessary search paths at early stages of the **BB** process.

According to our experimental experiences, however, it takes a significant amount of CPU run time (usually over six hours on a SPARC Sun4 SPARC station running the SUNOS 4.1.3 operating system) for a single experiment for $N \ge 40$ and $K \ge 30$, which makes collecting statistics difficult (although obtaining the best allocation for a single experiment is still possible). Some new techniques may be needed to reduce the search space. For example, based on the observation **P1**, one can co-allocate sequentially-executing modules in the TG subject to the same tight timing constraints. This can be done by calculating the release times and the latest completion times of all modules, while ignoring all IPC delays. If some module, M_i, has $LC_i - r_i$ equal to e_i[10] in all the component graphs of the TG, then this module and both its preceding and succeeding modules which are subject to the same timing constraint should be co-allocated.

unknown mean and variance.

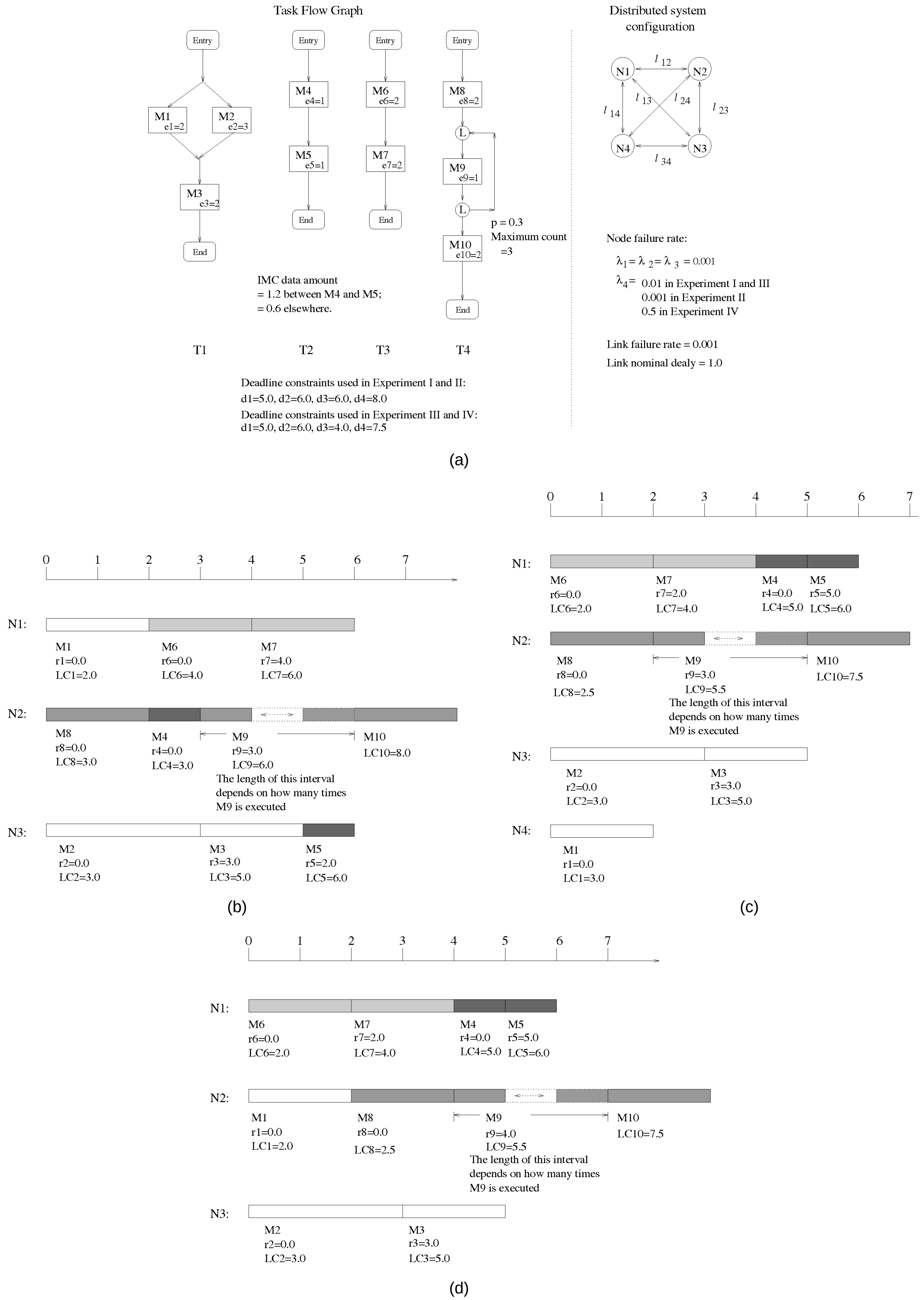
10. or, $LC_i - r_i \le f_m \cdot e_i$, where $f_m \ge 1.0$ is empirically determined if suboptimal allocations are allowed.

8 CONCLUSION

We have addressed the problem of allocating periodic task modules in a distributed real-time system subject to precedence constraints, timing requirements, and intermodule communications. The probability of no dynamic failure is used as the objective function to incorporate both the timeliness and logical correctness of real-time tasks/modules into module allocation. **MA** not only assigns modules to PNs, but also uses **MS** to schedule all modules assigned to each PN.

An interesting finding from the numerical experiments is that **MA** tends to allocate sequentially-executing modules subject to the same timing constraints to the same PN. Also, the common notion in general-purpose distributed systems that heavily communicating modules should be co-located [37], [23] may not always be applicable to real-time systems. Only in case there are enough resources to meet the timing requirement in a homogeneous distributed system, **MA** assigns modules so as to minimize IPCs. Based on a set of experiments using randomly-generated TGs and distributed systems, **MA** has also been shown to be computationally tractablefor $N \le 50$ and $K \le 40$.

Despite its advantages mentioned above, **MA** still takes a significant amount of time to locate an optimal allocation for the case of $N \ge 40$ and $K \ge 30$, due to the fact that there exist an extremely large number of search paths which might lead to an optimal solution and, thus, cannot be pruned at early stages of the **BB** process. One challenging extension to this research is to investigate the problem of grouping modules and/or PNs to reduce the size of the search space without resorting to a heuristic-directed technique. The conditions under which modules could be co-allocated, e.g., **P1** and **P2** observed in Section 7, are currently explored further, and will be reported in a forthcoming paper.



Fig, 6. An example showing how MA allocates modules: (a) The task graph and the system configuration used, (b) allocation and PN schedules for Experiment I: M_1, M_6, and M_7 are assigned to N_1; M_4, M_8, M_9, and M_{10} are assigned to N_2; M_2, M_3, and M_5 are assigned to N_3, (c) allocation and PN schedules for Experiment II and III: M_4, M_5, M_6, and M_7 are assigned to N_1; M_8, M_9, and M_{10} are assigned to N_2; M_2, and M_3 are assigned to N_3; and M_1 is assigned to N_4, (d) allocation and PN schedules for Experiment IV: M_4, M_5, M_6, and M_7 are assigned to N_1; M_1, M_8, M_9, and M_{10} are assigned to N_2; M_2, and M_3 are assigned to N_3.

LIST OF SYMBOLS

P_{ND}: the probability of no dynamic failure, i.e., the probability of all tasks making their deadlines.

P_{ND1}: the probability that all tasks within a planning cycle are completed before their deadlines.

P_{ND2}: the probability that all PNs are operational during the execution of modules assigned to them, and the communication links between communicating PNs are operational for all the intermodule communications that use these links.

N_T: the total number of periodic tasks in the system.

t_i: the invocation time of a periodic task T_i.

p_i: the period of a periodic task T_i.

d_i: the deadline of a periodic task T_i.

L: the planning cycle of a set of periodic tasks. It is computed as the least common multiple of $\{p_i : i = 1, 2, ..., N_T\}$.

$M_i \rightarrow M_j$: the precedence constraint imposed on modules M_i and M_j, meaning that the completion of M_i enables M_j to be ready for execution.

e_i: the execution time of a module M_i.

N: the number of modules to be allocated within a planning cycle.

K: the number of PNs available for module allocation.

x: a module allocation where $x_{ik} = 1$ if module M_i is assigned to PN N_k.

TG: the task flow graph representing the task system.

$TG(x)$: the set of modules which are already allocated under the allocation x. $TG(x) = TG$ if x is a complete allocation.

AN: the set of active nodes in the search tree which needs to be considered for node expansion in the next stage.

x_0: a null allocation which corresponds to the root of the search tree.

x_{opt}: an optimal allocation.

P^*_{ND}: the objective function value achieved by x_{opt}.

$\hat{P}_{ND}(x)$: the value by which the objective function, P_{ND}, of all child nodes expanded from the allocation x is upper-bounded.

NOTATION USED IN SECTIONS 4-7

$P_{tc}(T_\ell \mid x)$: the probability that a task T_ℓ is completed before its deadline under allocation x.

TG_c: a component task graph of TG.

$\{TG_c\}$: the set of component task graphs of TG.

p_c: the probability that TG is represented by TG_c.

$TG_c(x)$: the set of modules $\in TG_c$ which are allocated under x.

$S_k(x)$: the set of modules which are assigned to N_k under allocation x, i.e., $S_k(x) = \{M_i : x_{ik} = 1\}$.

r_i: the release time of module M_i which can be interpreted as the earliest time at which M_i can start its execution.

LC_i: the latest completion time of M_i. M_i must be completed before LC_i to ensure <u>all</u> tasks to meet their deadlines.

D_i: the critical time of M_i. M_i must be completed before D_i to ensure that the corresponding task (to which M_i belongs) will meet its deadline.

C_i: the completion time of M_i which is determined by **MSA**.

$\hat{e}_i$: the modified execution time of M_i. $\hat{e}_i$ is used to include the effect of queuing M_i on the release times of all those modules succeeding M_i.

$f_i(C_i)$: the cost incurred by completing M_i at time C_i.

$com_{ij}(x)$: the IMC time from M_i to M_j under allocation x.

d_{ij}: the IMC volume (measured in data units) from M_i to M_j.

$Y_{k\ell}$: the nominal delay (measured in time units per data unit) between two PNs, N_k and N_ℓ.

B: the minimal set of modules that are processed without any idle time from $r(B) = \min_{M_i \in B} r_i$ until $c(B) = r(B) + e(B)$, where $e(B) = \sum_{M_i \in B} e_i$.

b: the number of blocks in $S_k(x)$.

dg_i: the outdegree of M_i within a block under consideration.

$\hat{B}_i$: a subblock of $B - \{M_m\}$, where $1 \le i \le \hat{b}$, $\hat{b}$ is the number of subblocks in $B - \{M_m\}$, and M_m is the module scheduled to be executed if no other modules in B are waiting.

q_a: the looping-back probability of the loop L_a.

n_{L_a}: the maximum loop count of the loop L_a.

$q_{b,\ell}$: the branching probability of the ℓth branch of an OR-subgraph, O_b.

n_{O_b}: the number of branches in the OR-subgraph O_b.

$P_{tc}(T_\ell \mid TG_c, x)$: the probability that a task T_ℓ is completed before its deadline under allocation x for a given component task graph TG_c.

$\hat{T}_\ell \overset{\Delta}{=} \{M_i : M_i \in T_\ell \cap TG_c, dg_i = 0 \text{ with respect to } T_\ell \cap TG_c\}$: the set of modules without any successor in $T_\ell \cap TG_c$.

LP: the set of modules which are contained in loops.

OR: the set of modules which are on branches of OR-subgraphs.

λ_k: the *constant* exponential failure rate of N_k.

$\hat{\lambda}_{mn}$: the *constant* exponential failure rate of link ℓ_{mn}. We assume that λ_ks and $\hat{\lambda}_{mn}$s are statistically independent of one another.

t_{mn}: the nominal delay (measured in time units per data unit) of link ℓ_{mn}.

$n(k, \ell)$: the number of edge-disjoint paths from N_k to N_ℓ.

$I(m, n, k, \ell)$: the indicator variable such that $I(m, n, k, \ell) = 1$ if ℓ_{mn} lies on one of the $n(k, \ell)$ edge-disjoint paths from N_k to N_ℓ.

$R_{mn}(i, j, n_c, x)$: the probability that link ℓ_{mn} is operational during n_c occurrences of IMC between M_i and M_j under allocation x.

$R_{pn}(x)$: the probability that all PNs are operational during the execution of modules assigned to them under allocation x.

$R_{link}(x)$: the probability that all links are operational for performing all the IMCs that use them under allocation x.

LC_i^p: a pessimistic estimate of LC_i used in the branching process.

LC_i^o: an optimistic estimate of LC_i used in the branching process.

r_i^o: an optimistic estimate of r_i used in the branching process.

$\overset{\wedge}{PN}$: the set of PNs who need to reschedule the modules assigned to them because of the addition of $M_i \rightarrow N_k$ to a partial allocation.

ACKNOWLEDGMENTS

The work reported in this paper was supported in part by the U.S. Office of Naval Research under Grants N00014-92-J-1080 and N00014-91-J-1115 and by DARPA under Grant DABT63-95-C-0117. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the ONR or DARPA.

REFERENCES

[1] D.-T. Peng and K.G. Shin, "Modeling of Concurrent Task Execution in a Distributed System for Real-Time Control," *IEEE Trans.Computers*, vol. 36, no. 4, pp. 500-516, Apr. 1987.

[2] J.A. Stankovic, K. Ramamritham, and S. Chang, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems," *IEEE Trans. Computers*, vol. 34, no. 12, pp. 1,130-1,141, Dec. 1985.

[3] K. Ramamritham, J.A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *IEEE Trans. Computers*, vol. 38, no. 8, pp. 1,110-1,123, Aug. 1989.

[4] K.G. Shin and Y.-C. Chang, "Load Sharing in Distributed Real-Time Systems with State Change Broadcasts," *IEEE Trans. Computers*, vol. 38, no. 8, pp. 1,124-1,142, Aug. 1989.

[5] K.G. Shin and C.-J. Hou, "Analytic Models of Adaptive Load Sharing Schemes in Distributed Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 7, pp. 740-761, July 1993.

[6] C.-J. Hou and K.G. Shin, "Load Sharing with Consideration of Future Task Arrivals in Heterogeneous Distributed Real-Time Systems," *IEEE Trans. Computers*, vol. 44, no. 9, pp. 1,076-1,090, Sept. 1994.

[7] P.-Y. Ma, E.Y.S. Lee, and M. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems," *IEEE Trans. Computers*, vol. 31, no. 1, pp. 41-47, Jan. 1982.

[8] H.S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Trans. Software Eng.*, vol. 3, no. 1, pp. 85-93, Jan. 1977.

[9] V.M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," *IEEE Trans. Computers*, vol. 37, no. 11, pp. 1,384-1,397, Nov. 1988.

[10] C.E. Houstis, "Module Allocation of Real-Time Applications for Distributed Systems," *IEEE Trans. Software Eng.*, vol. 16, no. 7, pp. 699-709, July 1990.

[11] J.A. Bannister and K.S. Trivedi, "Task Allocation in Fault-Tolerant Distributed Systems," *Acta Informatica*, vol. 20, pp. 261-281, 1983.

[12] A.N. Tantawi and D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems," *J. ACM*, vol. 32, pp. 445-465, Apr. 1985.

[13] T.C.K. Chou and J.A. Abraham, "Load Balancing in Distributed Systems," *IEEE Trans. Software Eng.*, vol. 8, pp. 401-422, July 1982.

[14] C.C. Shen and W.H. Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion," *IEEE Trans. Computers*, vol. 34, no. 3, pp. 197-203, Mar. 1985.

[15] W.W. Chu and L.M.T. Lan, "Task Allocation and Precedence Relations for Distributed Real-Time Systems," *IEEE Trans. Computers*, vol. 36, no. 6, pp. 667-679, June 1987.

[16] W.W. Chu and K.K. Leung, "Module Replication and Assignment for Real-Time Distributed Processing Systems," *Proc. IEEE*, vol. 75, pp. 547-562, May 1987.

[17] D.-T. Peng and K.G. Shin, "Assignment and Scheduling of Communication Periodic Tasks in Distributed Real-Time Systems," *Proc. Ninth Int'l Conf. Distributed Computing Systems*, pp. 190-198, June 1989.

[18] S.M. Shatz and J.-P. Wang, "Model and Algorithm for Reliability-Oriented Task-Allocation in Redundant Distributed-Computer Systems," *IEEE Trans. Reliability*, vol. 38, pp. 16-27, Apr. 1989.

[19] S.M. Shatz, J.-P. Wang, and M. Goto, "Task Allocation for Maximizing Realiability of Distributed Computer Systems," *IEEE Trans. Computers*, vol. 41, no. 9, pp. 1,156-1,168, Sept. 1992.

[20] K.G. Shin, C.M. Krishna, and Y.H. Lee, "A Unified Method for Evaluating Real-Time Computer Controllers Its Application," *IEEE Trans. Automatic Control*, vol. 30, pp. 357-366, Apr. 1985.

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

[22] D. Fernández-Baca, "Allocating Modules to Processors in a Distributed System," *IEEE Trans. Software Eng.*, vol. 15, no. 11, pp. 1,427-1,436, Nov. 1989.

[23] K. Ramamritham, "Allocation and Scheduling of Complex Periodic Tasks," *IEEE Proc. 10th Int'l Conf. Distributed Computing Systems*, pp. 108-115, May 1990.

[24] C.-J. Hou and K.G. Shin, "Module Allocation with Timing and Precedence Constraints in Distributed Real-Time Systems," *IEEE Proc. 13th Real-Time Systems Symp.*, pp. 146-155, Dec. 1992.

[25] J.K. Strosnider and T.E. Marchok, "Responsive, Deterministic IEEE 802.5 Token Ring Scheduling," *J. Real-Time Systems*, vol. 1, pp. 133-158, Sept. 1989.

[26] B. Chen, G. Agrawal, and W. Zhao, "Optimal Synchronous Capacity Allocation for Hard Real-Time Communications with the Timed Token Protocol," *Proc. 13th Real-Time Systems Symp.*, pp. 198-207, Phoenix, Ariz., Dec. 1992.

[27] G. Agrawal, B. Chen, W. Zhao, and S. Davari, "Guaranteeing Synchronous Message Deadlines with the Timed Token Medium Access Control Protocol," *IEEE Trans. Computers*, vol. 43, no. 3, pp. 327-350, Mar. 1994.

[28] N. Malcolm and W. Zhao, "The Timed-Token Protocol for Real-Time Communications," *Computer*, vol. 27, no. 1, pp. 35-40, Jan. 1994.

[29] C.-C. Han, K.G. Shin, and C.-J. Hou, "On Non-Existence of Optimal Local Synchronous Bandwidth Allocation Schemes for the Timed-Token MAC Protocol," *Proc. IEEE 14th Int'l Phoenix Conf. Computers and Comm.*, pp. 191-197, Mar. 1995.

[30] C.-C. Han, K.G. Shin, and C.-J. Hou, "Synchronous Bandwidth Allocation for Real-Time Communications with the Timed-Token MAC Protocol," submitted for publication, Jan. 1997.

[31] C.-C. Han, C.-J. Hou, and K.G. Shin, "On Slot Allocation for Time-Constrained Messages in Dual-Bus Networks," *IEEE Trans. Computers*, vol. 46, no. 7, pp. 756-767, July 1997.

[32] D. Saha, M.C. Saksena, S. Mukherjee, and S.K. Tripathi, "On Guaranteed Delivery of Time-Critical Messages in DQDB," *Proc. IEEE INFOCOM '94, Conf. Computer Comm.*, vol. 1, pp. 272-279, June 1994.

[33] D. Ferrari and D.C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE J. Selected Areas in Comm.*, vol. 8, pp. 368-379, Apr. 1990.

[34] Q. Zheng and K.G. Shin, "On the Ability of Establishing Real-Time Channels in Point-to-Point Packet Switched Network," *IEEE Trans. Comm.*, vol. 42, Feb./Mar./Apr. 1994.

[35] W.L. Winston, *Operations Research, Applications and Algorithms*, second ed., chapter 7, pp. 321-326. PWS-KENT Publishing, 1987.

[36] K.R. Baker, E.L. Lawler, J.K. Lenstra, and A.H.G.R. Kan, "Preemptive Scheduling of a Single Machine to Minimize Maximum Cost Subject to Release Dates," *Operations Research*, pp. 381-386, Mar.-Apr. 1983.

[37] N.S. Bowen, C.N. Nikolaou, and A. Ghafoor, "On the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems," *IEEE Trans. Computers*, vol. 41, no. 3, Mar. 1992.

[38] C.-J. Hou, "Design and Analysis of Task Allocation and Re-Distribution in Distributed Real-Time Systems," PhD thesis, Univ. of Michigan-Ann Arbor, Dept. of Electrical Eng. and Computer Science, Aug. 1993. Also available as Technical Report CSE-TR-172-93, Univ. of Michigan-Ann Arbor.

Chao–Ju Hou received the BSE degree in electrical engineering in 1987 from National Taiwan University, the MSE degree in electrical engineering and computer science (EECS), the MSE degree in industrial and operations engineering, and the PhD degree in EECS, all from The University of Michigan, Ann Arbor, in 1989, 1991, and 1993, respectively. From August 1993 to July 1996, she was an assistant professor in the Department of Electrical and Computer Engineering at the University of Wisconsin–Madison. Since August 1996, she has been with the Department of Electrical Engineering at The Ohio State University–Columbus, where she is currently an assistant professor.

She is a recipient of the U.S. National Science Foundation CAREER award, Wisconsin/Hilldale Undergraduate/Faculty Research Fellowships, and Women in Science Initiative Awards in Wisconsin. Her research interests are in the areas of time-constrained communications, design and implementation of middleware services for QoS control and monitoring in high-speed networks, and performance modeling/evaluation. She has served on the program committees of several IEEE conferences, and is a member of IEEE Computer Society, ACM SIGCOMM, and Society of Woman Engineers.

Kang G. Shin received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and both the MS and PhD degrees in electrical engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. He is a 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 authored/coauthored more than 360 technical papers (about 150 of these in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. He has written (jointly with C.M. Krishna) a textbook *Real-Time Systems*, (McGraw-Hill, 1996). In 1987, he received the Outstanding *IEEE Transactions on Automatic Control* Paper Award for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award 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.

He has also been applying the basic research results of real-time computing to multimedia systems, intelligent transportation systems, and manufacturing applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes. (The latter is being pursued as a key thrust area of the newly-established NSF Engineering Research Center on Reconfigurable Machining Systems.)

From 1978 to 1982, he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at the University of California, Berkeley, and International Computer Science Institute, Berkeley, California, 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 in January 1991.

He is an IEEE fellow, 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 cochair for the 1992 International Conference on Parallel Processing, and served on 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 the *International Journal of Time-Critical Computing Systems*.