Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 80 (2007) 962–971
The Journal of Systems and Software
www.elsevier.com/locate/jss

# Resource management for real-time tasks in mobile robotics ☆

Huan Li [a,*,1], Krithi Ramamritham [b], Prashant Shenoy [a], Roderic A. Grupen [a], John D. Sweeney [a]

[a] Department of Computer Science, University of Massachusetts, 140 Governors Drive, Amherst, MA 01003, USA
[b] Department of Computer Science and Engineering, IIT Bombay, Powai, Mumbai 400076, India

Available online 16 November 2006

**Abstract**

Coordinated behavior of mobile robots is an important emerging application area. Different coordinated behaviors can be achieved by assigning sets of control tasks, or strategies, to robots in a team. These control tasks must be scheduled either locally on the robot or distributed across the team. An application may have many control strategies to dynamically choose from, although some may not be feasible, given limited resource and time availability. Thus, dynamic feasibility checking becomes important as the coordination between robots and the tasks that need to be performed evolves with time. This paper presents an on-line algorithm for finding a *feasible* strategy given a functionally equivalent set of strategies for achieving an application's goals.

We present two algorithms for feasibility improvement. Both consider communication cost and utilization bound to make resource allocation and scheduling decisions. Extensive experimental results show the effectiveness of the approaches, especially in resource-tight environments. We also demonstrate the application of our approach to real world scenarios involving teams of robots and show how feasibility analysis also allows the prediction of the scalability of the solution to large robot teams.


*Keywords:* Resource management; Task allocation; Distributed real-time systems

## 1. Introduction

Collaborating with one another to accomplish a common goal, for example, searching a burning building for trapped people, is a promising application for a team of robots. Human operators may direct the search by teleoperation, but wireless communications in these situations can be unreliable. When a search robot ventures outside a reliable communication range, a second robot can autonomously create a network to preserve quality of service between the operator and the search robot. One instantiation of such technology constructs a series, kinematic chain of mobile robots where each of them actively preserves Line-Of-Sight (LOS) (Sweeney et al., 2002) and intra-network bandwidth. In the simplest case, two pairwise coordinated controllers, *push* and *pull* are developed for a team of two robots.

The *push* and *pull* controllers differ in the way in which the LOS region is computed. If we call the pairwise robots as "Leader/Follower", the qualitative difference between the two configurations is that *pull* allows the leader robot to search for an area while "pulling" the following robot behind it; *push* allows the follower to specify the search area of the leader, in effect, "pushing" the leader along. The task models for these two strategies, namely, the *push* and *pull* controllers themselves, are depicted in Fig. 1. Here, each robot must run IR obstacle detection and odometric sensor processing tasks, denoted by $IR_i$ and $POS_i$, respectively. In addition, both robots must run a command processing task, $M_i$, which takes desired heading and speed commands and turns them into motor commands. All these tasks are preassigned to specific execution sites (robots). In the *push* configuration, the follower computes the LOS region in task $H_1$ (standing for the abbreviation

☆ This research was supported in part by DARPA SDR DABT63-99-1-0022 and MARS DABT63-99-1-0004, NSF grants CCR-0219520, EIA-0080119 and CNS-0323597.
* Corresponding author.
*E-mail address:* lihuan@cs.umass.edu (H. Li).
[1] Huan Li is now with the Beihang University, Beijing, China.

0164-1212/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2006.09.035

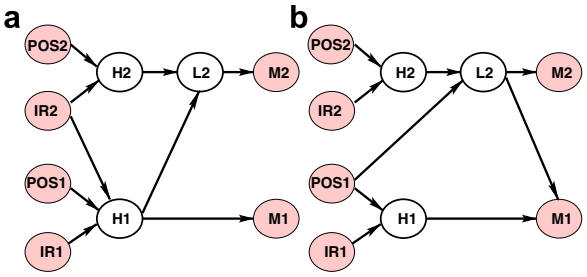


Fig. 1. Tasks in a *Leader/Follower* team. (a) *Push* strategy and (b) *Pull* strategy.

of the path planner), and passes it to task $L_2$, which computes a new movement vector of the leader that maximizes the search area while keeping the leader within the specified LOS region. In the *pull* configuration, the leader robot does the search, and, concurrently, computes the LOS region in task $L_2$.

In many cases, different combinations of *push* and *pull* controllers will, however, have the same coordinated search behavior. A discussion of how applications generate possible strategies is beyond the scope of this paper. Usually, applications determine the required type of coordinated behavior for a team of robots, and generate a set of functionally equivalent strategies. Each strategy consists of a set of periodic real-time tasks, e.g., sensor data must be transmitted/processed before they become invalid (which are indicated as deadlines). Since the control tasks in a strategy, such as $H_1$, $H_2$ and $L_2$ in the *push/pull* model, can be distributed among sites in a team, different task assignments may lead to different processor workloads and communication costs. Thus, a strategy that is valid at the application level may not always be *feasible* at the system level. One goal of this work is to find feasible, schedulable strategies from a set of functionally equivalent strategies given by the application.

One complicating aspect of this application domain is that the team is often moving and its size is not fixed. As robots enter or leave the team, the application must recompute the set of available strategies. Fig. 2 shows a sequence from a simulation with five robots using *push* and *pull* controllers, where robot 0 is the leader searching for the goal – the square in the lower left of the map. Each time the team changes, the application must run an *on-line* algorithm to determine the task allocation and schedulability of a new feasible strategy.

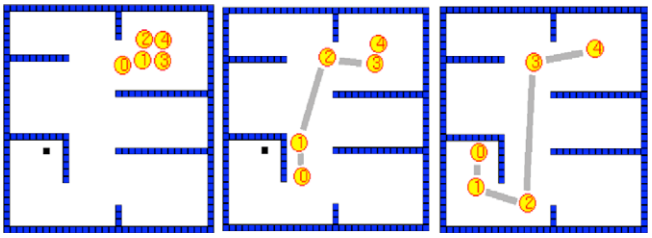

Fig. 2. A sequence of active robots in a robot team.

If two communicating tasks have to be allocated on different robots, communication over the shared communication medium happens. To avoid run time contention, the communication needs to be scheduled as well. Assigning tasks with precedence relationships in a distributed environment is in general an NP-hard problem (Peng et al., 1997), and even some of the simplest scheduling problems are NP-hard in the strong sense (Garey and Johnson, 1978). Given temporal and resource constraints, we propose two heuristic allocation algorithms. The purposes of those algorithms are: (1) minimizing communication overhead, and (2) minimizing and balancing the processor workload, so that the overall schedulability is improved and optimal coordinated behavior is achieved.

The contributions of this paper are as follows. We develop an on-line algorithm for finding a *feasible* control strategy, given a functionally equivalent set of strategies for achieving an application's goals. Specifically, we propose two simple but efficient algorithms for allocating control tasks to distributed processing entities. In order to improve schedulability, communication costs and utilization of processors are minimized. We have performed extensive evaluations for discovering the properties of the algorithms. We also exercise it using a case study of a real world example from mobile robotics to achieve a simple but efficient allocation and communication scheme for a team of robots. We believe that our approach can enable system developers to design efficient distributed embedded applications even though they possess a variety of temporal and resource constraints.

The rest of the paper is laid out as follows. In Section 2, the system model and goal are described. The details of the allocation and scheduling algorithms are provided in Section 3. Section 4 presents the results of evaluations from simulation. Section 5 analyzes a real-word robotic application. Section 6 discusses related work. Section 7 concludes the paper by summarizing the important characteristics of the algorithm and discusses future work.

## 2. System model and our goal

### 2.1. System and task model

A coordinated team consists of a set of sites (robots), each having an identical processor. In this paper, we use *site* and *processor* interchangeably. Robots in a team share a communication medium that allows broadcast communication between robots. A *strategy*, which is specified at the application level, is denoted at the system level by an acyclic *Task Graph* (TG), e.g., tasks graphs for *push/pull* strategies in Fig. 1. To accomplish a common goal for the team, a set of functionally equivalent strategies are supplied by applications.

In a TG, nodes represent tasks ($T_i$), directed edges between tasks represent communication relationships (*sender/receiver*) or precedence (*producer/consumer*) constraints. The amount of communication is denoted as a

communication cost attached to the edges. In our model, all tasks are periodic. Each task is characterized by a *period* $P_i$, *Worst Case Execution Time* (WCET) $C_i$, and relative *deadline* $D_i$, here, $D_i = P_i$. Periods can be *different* for different tasks. But if the sender and receiver run with arbitrary periods, task executions may get out of phase, which results in large latencies in communication (Saksena, 1998). Harmonicity constraints can simplify the reading/writing logic, reduce those latencies (Ryu and Hong, 1999) and increase the feasible processor utilization bound (Sha et al., 1994). To this end, we design the period of a receiver task as a multiple of the related sender's period.

### 2.2. Our goal

Given a set of sites and a set of functionally equivalent strategies, our goal is to find a feasible strategy. A strategy is *feasible* if and only if: (i) within the LCM (*L*east *C*ommon *M*ultiple) of task periods, each instance of a task is scheduled to run at its start time, and complete no later than its relative deadline; (ii) all constraints, such as *precedence*, are satisfied.

Based on the nature of the application, some tasks, e.g, sensor and motor systems, must run on specific processors, i.e., a particular robot platform. Other tasks, such as control or computation tasks, however, can be assigned to any site in the team. To find a feasible strategy, especially when the temporal and physical resources are tight, the system needs to:

1. assign unallocated tasks to appropriate processors so that the communication cost and the workload of each processor is minimized;
2. determine a feasible schedule for all task instances, including communication tasks.

The method for constructing communication tasks will be discussed in Section 3.2. In this paper, we assume the *Leader* robot is responsible for computing the feasibility of all available strategies and deciding which strategy the team should implement. The decision and execution process works as follows. The initial team settings are supplied by the application. At run time, the *Leader* determines feasibility and chooses a strategy for the team to execute. The *Leader* broadcasts this result to the rest of the team and waits for confirmation responses. (How these messages are propagated over the network to the rest of the team is beyond the scope of this paper.) After this broadcast phase is finished, the team members start the execution phase. We set a supervisory period, e.g., a multiple of the LCM, as the time interval during which the system runs under the current strategy. Once the supervisory period ends, the *Leader* checks if the team requires a new strategy due to a change in team size or topology, recomputes a new strategy, broadcasts it to the team, and a new execution phase begins.

## 3. Allocation and scheduling algorithms

We now give the details of the allocation and scheduling algorithms. The notation used in this paper is explained in Table 1. Here, $\mathrm{CCR}_{ij}$ is defined as:

$$\mathrm{CCR}_{i,j} = \frac{communication\_cost\ (T_i \rightarrow T_j)}{C_i + C_j}$$

### 3.1. Allocation algorithms

Optimal assignment of real-time tasks to distributed processors is an intractable problem. Two efficient algorithms, *Greedy* and *Aggressive* are proposed in this section. Based on the utilization of each processor and the amount of communication between tasks, both algorithms attempt to assign communicating tasks to the same processor with the purpose of minimizing total communication cost and thereafter, improving the schedulability. The difference between the two algorithms lies in whether the communication from the same processor is clustered or not. A dynamic incremental utilization threshold is used in both algorithms. The threshold aims to: (1) balance and minimize the workload of each processor; and (2) avoid violation of the utilization bound for schedulability purposes.

#### 3.1.1. Greedy algorithm

This algorithm takes into account the amount of communication and computation involved for *each pair* of communicating tasks. A decision is made as to whether these two tasks should be assigned to the same site, depending on the utilization condition; thereby, eliminating the communication cost. For schedulability purpose, a threshold $t$ is used to minimize and balance the utilization of each processor. Initially, $t$ is the maximum utilization value among all processors that have been loaded with sensor/motor tasks.

At each step, among all unallocated tasks, the algorithm selects the one that has the *largest* communication cost ratio (CCR), and then attempts to assign it to the same processor as its sender. For instance, if $\mathrm{CCR}_{i,j}$ has the largest value, where $T_i \rightarrow T_j$ and $T_i$ is located on site $S_x$, the algorithm will attempt to allocate $T_j$ to $S_x$, based upon

Table 1
Notation used in this paper

| Notation | Meaning |
|---|---|
| $T_i$ | Task ID |
| $C_i$ | Worst Case Execution Time (WCET) of task $T_i$ |
| $D_i^n$ | Deadline of the $n$th instance of $T_i$ |
| $E_i^n$ | Earliest start time of the $n$th instance of $T_i$ |
| $S_x$ | Site (Processor) ID |
| $u_x$ | Utilization of processor $S_x$ |
| $u_x^i$ | Utilization of $S_x$ that $T_i$ is on |
| $T_i \rightarrow T_j$ | Precedence constraint between $T_j$ and $T_i$ |
| $\mathrm{CCR}_{i,j}$ | Communication Cost Ratio for $T_i \rightarrow T_j$ |

whether or not the utilization of $S_x$ becomes larger than the threshold $t$. For example, let $t'$ denote the "expected" utilization of $S_x$ if $T_j$ is assigned to $S_x$. If $t' < t$, $T_j$ is allocated to $S_x$ and $t$ keeps the value. Otherwise, the algorithm will find a site $S_l$ that currently has the least utilization, and then attempts to assign $T_j$ to $S_l$. Two cases need to be discussed under this situation.

Case 1: $t' > 1$. In this case, if $S_l$ is different from $S_x$, and the new utilization $u'_l$ (after loading the selected task $T_j$) is less than *1*, $T_j$ is assigned to $S_l$ and the threshold is updated to the max value of $t$ and $u'_l$. Otherwise, no processor can load the task and the algorithm fails.

Case 2: $t' \leqslant 1$. In this case, we simply assign the task to the processor with least utilization, and update the threshold to $t'$. Now the new threshold indicates the new workload demands.

Depending on the threshold update result, if an appropriate processor is found, the algorithm moves on to the next step, using the new threshold. The algorithm is deemed successful if no task is left unallocated. However, if no site can be found for the selected task because any of the processor's utilization will be greater than 1 after loading this task, the algorithm fails. The pseudo-code for the *Greedy* allocation algorithm and the function of threshold update is shown in Tables 2 and 3, respectively.

Let $N$ be the number of tasks, $M$ be the number of processors. Generally there are $M^N$ allocation ways, and finding an optimal feasible allocation so that tasks meet all physical and temporal constraints is known to be NP-hard. In our algorithm, the *While* loop runs in $O(N^2)$ time. Consequently, the algorithm runs in $O(N^2)$ time.

### 3.1.2. Aggressive algorithm

Before we introduce a new algorithm, let us first look at an example depicted in Fig. 3, to illustrate the motivation. Here, $T_1$ and $T_2$ are preallocated on site $S_1$, $T_3$ is on site $S_2$; $T_4$, $T_5$ and $T_6$ are under consideration. The numbers attached to the arrows are communication costs (and CCRs). Worst case execution time ($C$) and period ($P$) of each task are also shown in the figure. For $S_1$ and $S_2$, the initial utilization values are: $u_1 = 0.45$, $u_2 = 0.5$, respectively.

According to *Greedy*, since $\mathrm{CCR}_{2,5}$ is the largest value, $T_5$ is considered. If assigning $T_5$ to $S_1$, the same site as $T_2$, the utilization will become $u_1 = 0.45 + 0.2 = 0.65 < 1$. Hence, $T_5$ is allocated to $S_1$. Now, let us consider $T_4$ and $T_6$. Because $\mathrm{CCR}_{3,4}$ has the largest value, task $T_4$ will be assigned to site $S_2$, the same site as $T_3$, and $u_2$ becomes 0.7. At this moment, no site can load task $T_6$ – the utilization will be larger than 1 if loading $T_6$, therefore, the algorithm finally fails.

The second allocation algorithm we propose takes into account the total communication cost, and selects the task that has the largest accumulated CCR to do assignment. For $T_4$ in above example, since the accumulated communication cost from $S_1$ is greater than that from $S_2$, i.e., $(\mathrm{CCR}_{1,4} + \mathrm{CCR}_{2,4}) > \mathrm{CCR}_{3,4}$, it is better to assign $T_4$ to $S_1$, other than $S_2$.

Because the utilization bound is still required for schedulability purpose, once the task is selected, the function of assignment and threshold update is the same as in *Greedy*. To this end, for the *Aggressive* algorithm, $R$ is the set of $(\sum_i \mathrm{CCR}_{i,j}, S_x, T_j)$, where $\forall T_i,\ T_i \in F,\ T_i \to T_j,\ Site(T_i) = S_x$. Line 6 in Table 2 is changed to: Insert $(\sum_i \mathrm{CCR}_{i,j}, S_x, T_j)$ to $R$; and line 13 is changed to: Delete $(\sum_i \mathrm{CCR}_{i,j}, S_x, T_j)$ from $R$; line 15 is changed to: Insert $(\sum_i \mathrm{CCR}_{i,k},$

Table 2
*Greedy* allocation algorithm

**Input:** a task graph $G = (E, V)$ with related periods, execution times and communication costs.
**Output:** a feasible task assignment
**Variables:** $F$: allocated task set, $I$: unallocated task set, $U$: set of utilization,
$t$: utilization threshold, $R$: set of $(\mathrm{CCR}_{i,j}, S_x, T_j)$, s.t. $T_i \to T_j,\ T_i \in F,\ Site(T_i) = S_x,\ T_j \in I$
**Algorithm 3.1:**

1. Initialize $U = \{u_i | i = 1, 2, \ldots, m\}$, that is for each processor $S_i$: $u_i = \sum_j \frac{C_j}{P_j}, T_j \in F \wedge Site(T_j) = S_i$;
2. Let $t = max(u_i), u_i \in U$; /* initialize the utilization threshold */
3. **If** $(t > 1)$,**do**
4. exit without solution;
5. **For** all $T_i \in F$ /* initialize R */
6. Insert $(\mathrm{CCR}_{i,j}, S_x, T_j)$ to $R$, s.t., $T_i \to T_j,\ Site(T_i) = S_x,\ T_j \in I$;
7. **While** ($I$ is not empty) **do**
8. Let $T_j$ be the task that has the largest value $\mathrm{CCR}_{i,j}$ out of $R$;
9. Let $u'_x = (u_x + \frac{C_j}{P_j})$; /* $T_i \to T_j, Site(T_i) = S_x$ */
10. **If** $((S_l = \textbf{thresholdUpdate}(u'_x, S_x, T_j)) < 0)$, **do**;
11. exit without solution; /* cannot find an appropriate processor */
12. Update set $F$, $I$ s.t., $F = F \cup \{T_j\}$, $I = I \setminus \{T_j\}$;
13. Delete $(\mathrm{CCR}_{i,j}, S_x, T_j)$ from $R$;
14. **For** all $T_k$ s.t., $T_j \to T_k$ and $T_k \in I$
15. Insert $(\mathrm{CCR}_{j,k}, S_l, T_k)$ to $R$; /* update R, $Site(T_j) = S_l$ */

Table 3
Threshold update

```
Processor thresholdUpdate(float t', Processor S_x, Task T_j)
/* return the allocation or −1 if fails, T_j and processor S_x is selected,
   t' = u'_x */
 1. Case 1: t' ⩽ t, do /* t' is less than the threshold t */
 2.    Assign task T_j to processor S_x;
 3.    Update U with the new utilization u_x = t';
 4.    Return S_x;
 5. Case 2: t' > t, do /* t' is larger than the threshold t */
 6.    Find S_l that has the least utilization u_l = min(u_i), u_i ∈ U,
 7.    Let u'_l = u_l + C_j/P_j;
 8.    Case 2.1: t' > 1, do /* processor S_x cannot load T_j */
 9.       If (S_l ≠ S_x) ∧ (u'_l ⩽ 1), do
10.          Allocate task T_j to processor S_l;
11.          Update U with u_l = u'_l;
12.          t = max(t, u'_l);
13.          Return S_l;
14.       Else
15.          Return −1; /* cannot find an assignment for T_j */
16.    Case 2.2: t' ⩽ 1, do /* u'_l ⩽ t' ⩽ 1 */
17.       Allocate task T_j to processor S_l;
18.       Update U with u_l = u'_l;
19.       t = t';
20.       Return S_l;
```

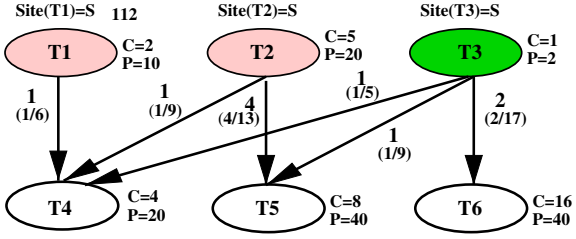


Fig. 3. A simple task graph example.

$S_l, T_k)$ to $R$, where $\forall T_i$, $Site(T_i) = S_l$, $T_i \rightarrow T_k$. Following *Aggressive*, when $T_4$ is considered, it will be assigned to $S_1$ and thereafter $u_1 = 0.65 + 0.2 = 0.85$. Now, if we assign $T_6$ to site $S_2$, the utilization of $S_2$ becomes 0.9, and the algorithm succeeds. This slightly more complicated algorithm is shown to be more effective in the sense of higher schedulability in Section 4.

### 3.2. Making scheduling decisions

After a successful task assignment is found, we need to find a feasible schedule for all instances of the tasks. Searching for a feasible schedule for real-time tasks subject to precedence constraints in a distributed environment is an intractable problem in the worst case, therefore, we propose to use heuristic methods. Before we discuss the approach, first, let us define some terminology.

*Earliest start time*: The earliest start time of an instance of a task is derived from the *precedence* constraints. Let $L$ be the LCM of task periods. If task $T_i$ has no predecessors, the first instance is ready to execute at time 0, denoted as $E_i^1 = 0$; and for the $n$th instance of that task, $E_i^n = (n-1) \times P_i$, where $1 \leqslant n \leqslant L/P_i$. If $T_i$ has predecessors, its first instance becomes enabled only when all its predecessors have completed execution. In order to achieve this condition, the tasks in the original task graph are topologically ordered. When a task $T_i$ is processed, the lower bound of $E_i^1$ is set to $max(E_i^1, E_k^1 + C_k)$, where $\forall k$, $T_k \in Predecessors(T_i)$. Since we will model communication as a task if two communicating tasks are on different sites, and we have harmonicity constraints for all such pairs, initially, the lower bound of $E_i^n$ is assigned to $(n-1) \times P_i + E_i^1$.

*Communication task*: If the pair of communicating tasks have been assigned to the same site, the communication cost is avoided; otherwise, communication needs to be scheduled. Consider $T_i \rightarrow T_j$ and $m = \frac{P_j}{P_i}$, we will construct $m$ communication tasks; and the $k$th such task $T_{\text{comm}}^k$ has the following features.

1. $P_{\text{comm}}^k = P_j$. $T_j$ needs to process data sent from one instance of $T_i$ only once during one period of $T_j$;
2. $D_{\text{comm}}^k = P_{\text{comm}}^k - C_j$. This is an upper bound since the communication should finish its execution no later than the latest start time of $T_j$;
3. $E_{\text{comm}}^k = E_i^k + C_i$. This is a lower bound because the communication task should begin execution at least after the completion of the related instance of $T_i$.

The data sent by the communication tasks for the same sender task are buffered at the destination site until the receiver task begin to process them. In this paper, we assume the transmission is lossless once it is scheduled.

Given a task graph with location information, the search algorithm attempts to determine a feasible schedule for all task instances and communication tasks. It starts with an empty partial schedule as the root and tries to extend the schedule with one more task by moving to one of the vertices at the next level in the search tree. It continues this process until a feasible schedule has been found. The heuristic function $H$ is applied to each of the remaining unscheduled tasks at each level of the tree. The task with the smallest value is selected to extend the current partial schedule. Once a task is scheduled, the earliest start times of all its successors are updated accordingly. Currently, the potential heuristic functions we use are: (1) Earliest deadline first: $H(T) = Min\_D$; (2) Minimum earliest start time first: $H(T) = Min\_E$; (3) Minimum laxity first: $H(T) = Min\_L = min(D_i - (E_i + C_i))$; (4) $H(T) = Min\_D + W \times Min\_E$; (5) $H(T) = Min\_D + W \times Min\_L$; (6) $H(T) = Min\_E + W \times Min\_L$. The parameter $W$ is the weight factor used to adjust the effect of different temporal properties of the tasks.

## 4. Simulation results

We conducted several experiments to study the features of the proposed algorithms, How these algorithms can be applied to an actual robotics application is discussed in Section 5. Tasks generated in a directed acyclic graph have the following characteristics.

The computation time $C_i$ of each task $T_i$ is uniformly distributed between $C_{min}$ and $C_{max}$ set to 10 and 60 time units, respectively. The communication cost lies in the range $(CR \times C_{min}, CR \times C_{max})$, where CR is called *Communication Ratio* and are set between 0.1 and 0.4.

To address harmonicity relationships, we set a period range, $(minP_i^I, maxP_i^I)$, for each input task $T_i$ (task without incoming edges), and $(1, maxP_j^O)$ for each output task $T_j$ (task without outgoing edges), where $minP_i^I = Lower \times C_i$ and $maxP_i^I = Upper \times C_i$, $Lower = 1.1$ and $Upper = 4.0$. To ensure that the periods of output tasks are no less than those of input tasks, a parameter, *mult_factor* is used to set the upper bound of the period for output task $T_j$: $maxP_j^O = mult\_factor \times max(maxP_i^I)$, where $T_i$ are input tasks and *mult_factor* is randomly chosen between 1 and 5. In order to make periods harmonic, first, we process input tasks and make their periods harmonic; then we tailor the techniques from Ryu and Hong (1999) to process output tasks; finally, we use the GCD technique for intermediate tasks to achieve harmonicity constraints. $P_1^O$, which is the smallest period of all output tasks, is calculated upon the largest period of input tasks $(P_m^I)$, $P_1^O = \lfloor maxP_1^O / P_m^I \rfloor \times P_m^I$. Other output tasks' periods are computed upon $P_1^O$ to achieve harmonicity.

All the simulation results shown in this section are obtained from the average value of 10 simulation runs. For each run, we generate 100 test sets, each set satisfying $\sum_{i=1}^{n}(C_i/P_i) \leqslant m$, where $n$ is the number of tasks and $m$ is the number of processors. For a given task set, if this condition does not hold, at least one processor utilization will be larger than 1. Obviously, this does not eliminate all infeasible task sets because the presence of communication costs are not considered. However, since feasibility determination is intractable, if one heuristic scheme is able to determine a feasible schedule while another cannot, we can conclude that the former is superior. We use *Success Ratio*(SR) to compare the performance:

$$SR = \frac{NT^{succ}}{NT}$$

Here, $NT^{succ}$ is the total number of schedulable task sets found by the algorithm, and NT is the total number of task sets tested. In this paper, $SR = \left(\sum_{i=1}^{10} SR_i\right)/10$, where $SR_i = NT_i^{succ}/100$. The tests involved a system with 2 to 12 processors. Resources other than CPUs and the communication network are not considered.

### 4.1. Selecting a scheduling heuristic

In order to eliminate the bias from scheduling heuristic functions when study the performance of allocation algorithms, we first investigate the scheduling heuristics.

For both *Greedy* and *Aggressive* algorithms, we find *Min_E* is the best simple scheduling heuristic, while $Min\_D + W{*}Min\_E$ has substantially better performance than other heuristics including *Min_E*. This is because *earliest start time* of each instance of a task encodes the basic *precedence* information, and another important factor, *deadline*, is also taken into account in $Min\_D + W * Min\_E$. Fig. 4(a) shows the effect of different scheduling heuristic functions when using *Greedy* algorithm. We have similar results for *aggressive* algorithm.

Since $Min\_D + W * Min\_E$ is a weighted combination of simple heuristics, we also investigate its sensitivity to the weight ($W$) values for various number of processors. When the weight changes from 0 to 4 (or to 12 if the number of processors is 2), we see a significant performance improvement. Because the performance is only slightly affected when the weight changes from 4 to 30 (or 12 to 30 if the number of processors is 2), we will choose $W = 4$ for the following experiments.

### 4.2. Performance of allocation algorithms

In this section, we evaluate the performance of three allocation algorithms: *Greedy*, *Aggressive* and *Random*. The *Random* scheme will randomly assign an unallocated task to a processor as long as the utilization is less than 1. Fig. 5 illustrates the results when communication ratio

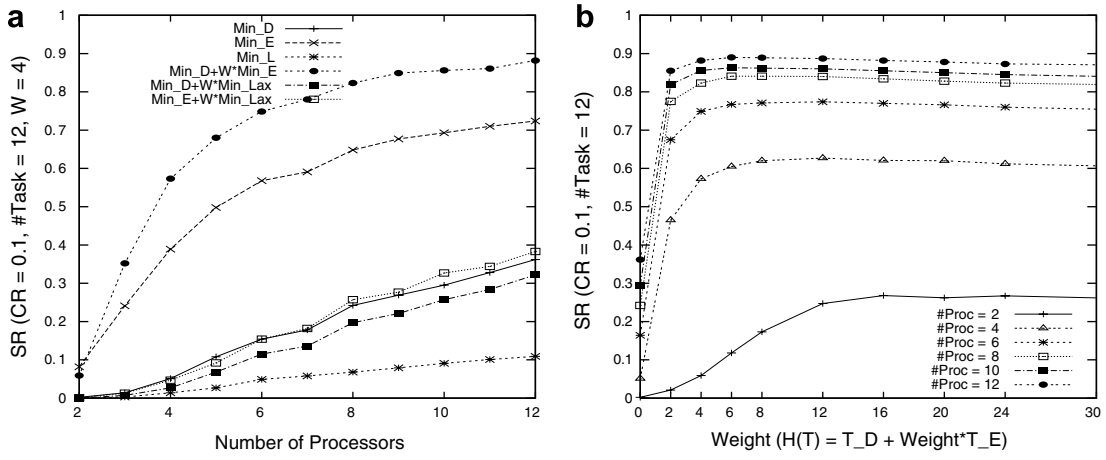


Fig. 4. Effect of scheduling heuristic and weight (*Greedy*).

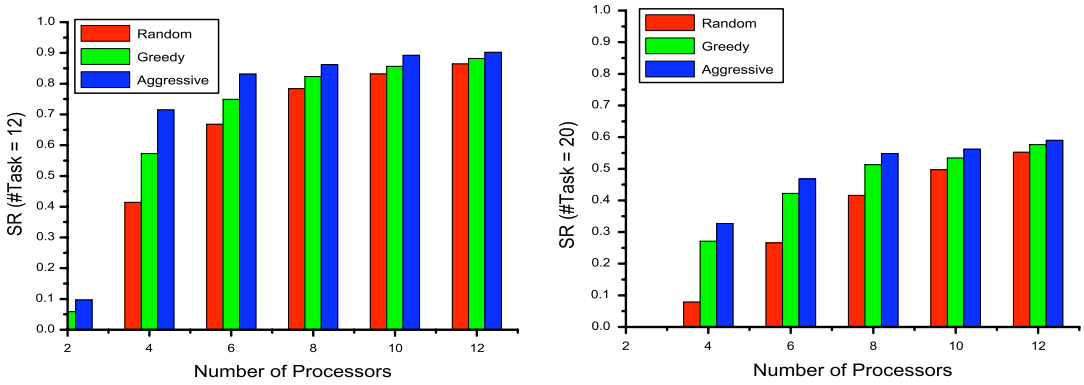


Fig. 5. Performance of allocation algorithms (CR = 0.1).

is set to 0.1 (we have similar results for CR = 0.4). As shown in the graphs, for each instance with different task set size, *Aggressive* outperforms *Greedy*, and *Greedy* outperforms *random*. The gains come from two factors: (1) the elimination of communication cost, (2) minimizing utilization for each processor. Since *Greedy* only considers the individual communication cost at each step, while *Aggressive* clusters and eliminates communication costs as many as possible, it is not surprising that *Aggressive* achieves better performance than *Greedy*.

The other observation is when the communication cost is heavier, the improvement in performance of *Greedy* or *Aggressive* is larger. Table 4 shows the difference in improvement of *Greedy* and *Aggressive* over *Random*. For both *Greedy* and *Aggressive*, in most cases, the improvements with CR = 0.4 are much greater than those with CR = 0.1. When CR = 0.4, the communication introduces more workload, and therefore, the system has more resource contention in terms of utilization boundary and deadline guarantee. So communication costs dictate the schedulability much more than the case when CR = 0.1. In contrast to random assignment, our approaches exploit this important property to direct the allocation assignment, hence, they work better in the resource-tight environment.

Finally, we find as the number of processors increases, the improvement for both *Greedy* and *Aggressive* tends to decrease for a given task set. This result further demonstrates the tighter the resource, the better our algorithms perform.

Table 4
Improvement of *Greedy* and *Aggressive* over *Random* (Percentage)

| # Proc. | 4 | 6 | 8 | 10 | 12 |
|---|---|---|---|---|---|
| (a) *Greedy over Random* | | | | | |
| CR = 0.1 | 19.2 | 15.6 | 9.7 | 3.7 | 2.4 |
| CR = 0.4 | 17.9 | 14.6 | 15.0 | 13.7 | 15 |
| (b) *Aggressive over Random* | | | | | |
| CR = 0.1 | 24.8 | 20.2 | 13.2 | 6.5 | 3.8 |
| CR = 0.4 | 22.8 | 18.3 | 18.1 | 16.1 | 16.3 |

### 4.3. Effect of communication

Allocation approaches attempt to assign communicating tasks to the same site to minimize the total communication cost. However, in cases where such tasks have to be placed on different sites, the communication cost becomes a very important factor in overall performance. To investigate the effect of communication, we compare the results with CR = 0.1 and CR = 0.4 by varying the number of processors and the number of tasks. The results are illustrated in Fig. 6.

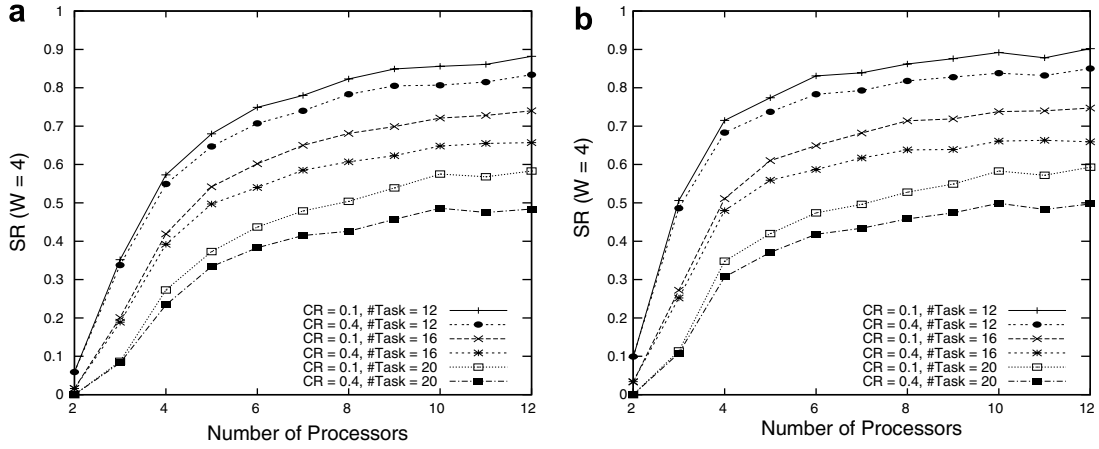


Fig. 6. Effect of communication. (a) Greedy and (b) aggressive.

Our results show that when the number of processors is very limited, e.g., 2 or 3, the performance is almost the same. This is because in such a situation, it is hard to find a feasible schedule for both cases. But as the number of processors increases, the performance for CR = 0.1 is better than that for CR = 0.4. This is because each communication introduces extra workload in addition to the precedence constraints, if the communicating tasks have to be assigned to different processors. When the communication ratio (CR) is set to be larger, the communication costs are bigger, which have more impact on the earliest start time of related consumers and the overall system.

## 5. Application of our algorithms to mobile robotics

In this section, we return to the robotic problem discussed in Section 1, where two strategies, *push* and *pull*, are given for a team of two robots. In Table 5, the WCET of tasks are taken from an experimental implementation on a StrongARM 206 MHz CPU; in Table 6, communication costs are based on the bytes transmitted using 802.11b wireless protocol with 11 Mbit/s transmission rate. Although 802.11b does not allow for real-time transmission guarantees, by prescheduling communications, medium contention is avoided. The periods are assigned with 220 ms for all sensor tasks, motor drivers and controller tasks by the application. Although these figures are given for tasks in Fig. 1, they are compatible to tasks that occur with more robots. Let us consider the scenario when a third robot wants to join the team. Since the *push* and *pull* controllers are pairwise, there are four possible strategies.

First, let us use the *Aggressive* algorithm to analyze the task assignment in each strategy. In this example, since the accumulated communication cost is considered, the allocation is the same for all strategies: $H_1$ is assigned to $S_1$, $H_2$ to $S_2$, $H_3$ to $S_3$, $L_2$ to $S_2$ and $L_3$ to $S_3$.

Next, the algorithm will see which strategies are schedulable under the heuristic $Min\_D + W \times Min\_E$. To simplify the analysis, here $W$ is set to 1. The completion times for tasks on each site are shown in Table 7. The algorithm finds that, except for {*Push*, *Pull*}, denoted as {*ph*,*pl*}, all other strategies are feasible, but with different completion times (including communication delay). Since multiple strategies are feasible, the application can use some criteria to rank the strategies. In this case, if the total laxity is used as the criterion, the application will choose the {*Pull*, *Push*} strategy, because it has the maximum value.

The application can then use the feasible results when computing new sets of strategies. For example, if at some time a fourth robot joins the team, the application immediately knows that any strategy that contains {*Push*, *Pull*} will not be feasible, since that strategy was already determined to be infeasible. Therefore, the application can use the feasibility analysis to prune infeasible strategies as the team's size scales. The idea of using the proposed algorithm to do the scalability analysis is shown in Sweeney et al. (2003).

Table 5
WCETs(ms) of tasks in Fig. 1

| Task | IR1(2) | *Pos*1(2) | $H_1$ | $H_2$ | $L_2$ | *M*1(2) |
|---|---|---|---|---|---|---|
| Push | 20 | 120 | 35 | 25 | 5 | 20 |
| Pull | 20 | 120 | 25 | 25 | 18 | 20 |

Table 6
Communication costs for Fig. 1

| | $IR_{1(2)} \rightarrow H_{1(2)}$ | $Pos_{1(2)} \rightarrow H_{1(2)}, L_2$ | $H_{1(2)} \rightarrow L_2$ | $H_1 \rightarrow M_1$ | $L_2 \rightarrow M_{2(1)}$ |
|---|---|---|---|---|---|
| *Push* | 0.02327 | 0.01236, 0 | 2.979 | 2.979 | 2.979(0) |
| *Pull* | 0.02327 | 0.01236 | 0(2.979) | 2.979 | 2.979(2.979) |

Table 7
The completion time for all strategies

| Strategy | {*Ph*, *Ph*} | {*Pl*, *Pl*} | {*Ph*, *Pl*} | {*Pl*, *Ph*} |
|---|---|---|---|---|
| Site 1 | 195 | 205.979 | 222.979 | 195 |
| Site 2 | 202.979 | 208.958 | 220 | 205.979 |
| Site 3 | 210.958 | 203 | 230.958 | 203 |

## 6. Related work

Numerous research results have demonstrated the complexity of design for real-time system, especially with respect to temporal constraints (Gerber et al., 1995; Ramamritham, 1996; Ryu and Hong, 1999; Saksena, 1998; Saksena and Hong, 1996). Also the schedulability analysis for distributed real-time systems has attracted a lot of attention in recent years (Kim et al., 2000; Palencia and Harbour, 1998; Palencia and Harbour, 1999; Wang and Farber, 1999). For tasks with temporal constraints, researchers have focused on generating task attributes, e.g., period, deadline and phase. For example, Gerber et al. (1995) and Saksena and Hong (1996) proposed the period calibration technique to derive periods and related deadlines and release times from given end-to-end constraints. Techniques for deriving system-level constraints from performance requirements are proposed by Seto et al. (1998, 1996). When end-to-end constraints are transformed into intermediate task constraints, most previous research results are based on the assumption that task allocation has been done *a priori*. However, *schedulability* is clearly affected by both the temporal characteristics and the allocation of tasks. A more comprehensive approach that takes into account the task temporal characteristics and allocations, in conjunction with schedulability analysis, is required.

For a set of *independent* periodic tasks, Liu and Layland (1973) first developed the feasible workload condition for

schedulability analysis under uniprocessor environments. Much later, Baruah et al. (1996) presented necessary and sufficient conditions, namely, $U \leqslant n$ ($n$ is the number of processors) based on *P-fairness* scheduling for multiprocessors. Also, the upper bounds on workload specified for the given schedules, e.g., EDF and RMA, are derived for homogeneous or heterogeneous multiprocessor environments (Andersson et al., 2001; Baruah, 2001; Funk et al., 2001; Goossens et al., 2002; Goossens et al., 2003; Srinivasan and Baruah, 2002). All these techniques are for preemptive tasks and task or job migrations are assumed to be permitted without any penalty. If *precedence* and *communication* constraints exist, these results cannot be directly used.

With regards to the distributed environment, Tindell et al. (1992), Peng et al. (1997), Abdelzaher and Shin (2000) and Ramamritham (1995) studied the task allocation and scheduling problem. In their models, tasks can have precedence or communication constraints. From this perspective, their work comes closest to ours. Tindell et al. describes an approach to solving the task allocation problem using a technique known as *simulated annealing*. In their work, simulated annealing is proven to be an effective approach to task allocation. However, it may not be directly used as an on-line algorithm, considering the speed of the algorithm. Besides, well-balanced allocations are shown in their paper to result in infeasible solutions, since a token protocol is used as the message transmission model, and a high bus utilization gives a high token rotation time, resulting in less schedulable solutions. Our algorithm, however, achieves well-balanced allocation and minimal message transmission at the same time, giving rise to the schedulability improvement. By using a branch-and-bound search algorithm (Peng et al., 1997), the optimal solution, in the sense of minimizing maximum normalized task response time, to the problem of allocating communicating periodic tasks to heterogeneous processing nodes is found. Though the heuristic guides the algorithm efficiently toward an optimal solution, the algorithm cannot be simply applied and extended to our environment. The major differences are: (1) applications require that the decision be made on-line; (2) we consider a non-preemptive schedule which is NP-hard in the strong sense even without precedence constraints (Garey and Johnson, 1979), while the algorithm (Baker et al., 1983) used in their method is for finding a preemptive schedule; and (3) the precedence constraints are predetermined among specific instances of tasks in their algorithm, while in our approach, this is accomplished by the scheduling subject to the precedence constraints. In Abdelzaher and Shin (2000), a period-based method is proposed to the problem of load partitioning and assignment for large distributed real-time application. Scalability is achieved by utilizing a recursive divide-and-conquer technique. Ramamritham (1995) discussed a static algorithm for allocating and scheduling components of periodic tasks across sites in distributed systems. How to allocate replicated tasks is a major issue addresses in the algorithm. Our task allocation and scheduling algorithm, however, focuses on the improvement of schedulability by: (1) using a dynamic increasing threshold to bound the utilization along with each allocation step; and (2) considers the *precedence* constraints as soon as possible by setting the earliest start time into the heuristic scheduling function.

## 7. Conclusion and future direction

Allocating and scheduling of real-time tasks in a distributed environment is a difficult problem. The algorithms discussed in this paper provide a framework for allocating and scheduling periodic tasks with precedence and communication constraints in a distributed dynamic environment, such as a mobile robot system.

Our algorithm was applied to a real world example from mobile robotics to achieve a simple but efficient allocation and scheduling scheme for a team of robots. We believe that this approach can enable system developers to design a predictable distributed embedded system, even if there are a variety of temporal and resource constraints.

Now we discuss some of the possible extensions to the algorithm. First, if the system design does not have pre-allocated tasks, the heuristic is still applicable. In this case, the initial threshold is 0. After selecting the first pair of communicating tasks and randomly assigning them to a processor, the algorithm can continue to work on remaining tasks as discussed in the original algorithms.

Second, the algorithm can be tailored to apply to heterogeneous systems. If processors are not identical, the execution time of a task could be different if it runs on different sites. To apply our approach in such an environment, first, we can take the worst case communication cost ratio, which is calculated by the slowest processors for each pair of communicating tasks, and then we can use these values as estimates to choose the task to be considered next. Second, when we select the processor, if the task can be assigned to the processor that the producer is on, then we are done; otherwise, we need to consider the utilization and the speed of a processor the same time, e.g., compare the utilization from the fastest processors to see which processor will have the least utilization after loading the task, and choose the one with the minimum value. After assigning each task, the threshold will change in a way similar to the original algorithm.

## References

Abdelzaher, T.F., Shin, K.G., 2000. Period-based load partitioning and assignment for large real-time applications. IEEE Transactions on Computers 49 (1), 81–87.

Andersson, B., Baruah, S., Jonsson, J., 2001. Static-priority scheduling on multiprocessors. In: IEEE real-time systems symposium, December 2001. pp. 193–202.

Baker, K.R., Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R., 1983. Preemptive scheduling of a single machine to minimize maximum cost to release dates and precedence constraints. Operations Research 31 (2), 381–386.

Baruah, S., 2001. Scheduling periodic tasks on uniform multiprocessors. Information Processing Letters 80 (2), 97–104.

Baruah, S.K., Cohen, N.K., Plaxton, C.G., Varvel, D.A., 1996. Proportionate progress: a notion of fairness in resource allocation. Algorithmica 15 (2), 600–625.

Funk, S., Goossens, J., Baruah, S., 2001. On-line scheduling on uniform multiprocessors. In: IEEE real-time systems symposium, December 2001. pp. 183–192.

Garey, M.R., Johnson, D.S., 1978. Strong np-completeness results: motivation, examples, and implications. ACM 25 (3), 499–508.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability. W.H.Freeman And Company, New York.

Gerber, R., Hong, S., Saksena, M., 1995. Guaranteeing real-time requirements with resource-based calibration of periodic processes. IEEE Transactions on Software Engineering 21 (7), 579–592.

Goossens, J., Funk, S., Baruah, S., 2002. Edf scheduling on multiprocessor platforms: some(perhaps)counterintuitive observations. In: Real-Time Computing Systems and Applications Symposium, March 2002.

Goossens, J., Funk, S., Baruah, S., 2003. Priority-driven scheduling of periodic task systems on multiprocessors. Real-Time Systems 25 (2–3), 187–205.

Kim, T., Lee, J., Shin, H., Chang, N., 2000. Best case response time analysis for improved schedulability analysis of distributed real-time tasks. In: Proceedings of ICDCS workshops on Distributed Real-Time systems, April 2000.

Liu, C.L., Layland, J.W., 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. ACM 20 (1), 46–61.

Palencia, J.C., Harbour, M.G., 1998. Schedulability analysis for tasks with static and dynamic offsets. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, December 1998.

Palencia, J.C., Harbour, M.G., 1999. Exploiting preceding relations in the schedulability analysis of distributed real-time systems. In: Proceedings of the 20th IEEE Real-Time Systems Symposium, December 1999.

Peng, D., Shin, K.G., Abdelzaher, T.F., 1997. Assignment and scheduling communicating periodic tasks in distributed real-time systems. IEEE Transactions on Software Engineering 23 (12).

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

Ramamritham, K., 1996. Where do time constraints come from and where do they go? International Journal of Database Management 7 (2), 4–10.

Ryu, M., Hong, S., 1999. A period assignment algorithm for real-time system design. In: Proceedings of 1999 Conference on Tools and Algorithms for the Construction and Analysis of System.

Saksena, M., 1998. Real-time system design: A temporal perspective. In: Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering, May 1998. pp. 405–408.

Saksena, M., Hong, S., 1996. Resource conscious design of distributed real-time systems an end-to-end approach. In: Proceedings of 1999 IEEE International Conference on Engineering of Complex Computer Systems, October 1996. pp. 306–313.

Seto, D., Lehoczky, J.P., Sha, L., Shin, K.G., 1996. On task schedulability in real-time control system. In IEEE real-time systems symposium, December 1996. pp. 13–21.

Seto, D., Lehoczky, J.P., Sha, L., 1998. Task period selection and schedulability in real-time systems. In IEEE real-time systems symposium, December 1998. pp. 188–198.

Sha, L., Rajkumar, R., Sathaye, S.S., 1994. Generalized rate monotonic scheduling theory: a framework for developing real-time systems. Proceedings of the IEEE 82 (1), 68–82.

Srinivasan, A., Baruah, S.K., 2002. Deadline-based scheduling of periodic task systems on multiprocessors. Information Processing Letters 84 (2), 93–98.

Sweeney, J., Brunette, T., Yang, Y., Grupen, R., 2002. Coordinated teams of reactive mobile platforms. In: Proceedings of the 2002 IEEE Conference on Robotics and Automation, Washington, DC, May 2002.

Sweeney, J., Li, H., Grupen, R., Ramamritham, K., 2003. Scalability and schedulability in large, coordinated, distributed robot systems. In: Proceedings of the 2003 IEEE Conference on Robotics and Automation, May 2003, Taipei, Taiwan.

Tindell, K., Burns, A., Wellings, A., 1992. Allocating hard real-time tasks: an NP-hard problem made easy. Real-Time Systems 4, 145–165.

Wang, S., Farber, G., 1999. On the schedulability analysis for distributed real-time systems. In: Joint IFAC–IFIP WRTP'99 & ARTDB-99, May 1999.