Real-Time Support for Mobile Robotics *

Huan Li, John Sweeney, Krithi Ramamritham, Roderic Grupen, and Prashant Shenoy
Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003
{*lihuan,sweeney,krithi,grupen,shenoy*}*@cs.umass.edu*

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 online algorithm for finding a feasible *strategy given a functionally equivalent set of strategies for achieving an application's goals.*

We present two heuristics for feasibility checking. Both consider communication cost and utilization bound to make allocation (of tasks to execution sites) 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.

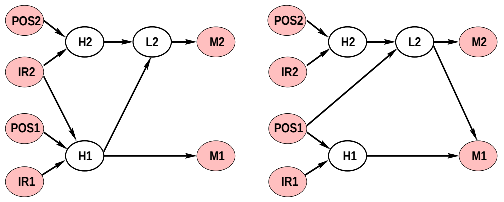
Keywords: *Distributed real-time systems, allocation, schedulability, precedence constraint*

1 Introduction

A promising application for a team of mobile robots is to collaborate with each other to accomplish a common goal, for example, searching a burning building for trapped people. 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. The lead robot can thus penetrate further into the rubble at the expense of communication latencies and distributed control overhead. One instantiation of such a strategy constructs a series, kinematic chain of mobile robots where each of them actively preserves Line-Of-Sight (LOS) [27] and intra-network bandwidth. In the simplest case, pairwise coordinated controllers were developed for a team of two robots. One controller, denoted *pull*, allows a leader robot to search an area while "pulling" a following robot behind it. The other controller, *push*, allows a follower to specify the search area of the leader, in effect, "pushing" the leader along. The application constructs a strategy by assigning *push* or *pull* controllers to the entire team. The task models for these two strategies, namely, the *push* and *pull* controllers themselves, are shown in Figure 1. In the two task graphs, sensor and motor tasks IR_i, POS_i, and M_i are preassigned to specific execution sites, while three control tasks H_1, H_2, and L_2 may reside on either team member, if necessary, to optimize processor utilization or communication costs. The H_1 and H_2 tasks are used by the robots to determine current search and LOS areas, respectively, while L_2 is used for coordinating the desired movements of the robots so that LOS is maintained and the search can make progress. The functionality of the team is not affected by changing the allocations of the control tasks. The differences between *push* and *pull* can be seen at the task graph level. For example, with *push*, the communication $H_1 \rightarrow L_2$ specifies to the leader which areas it may search, while with *pull*, $L_2 \rightarrow M_1$ tells the follower where it may move.

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 n robots, and generate a set of functionally equivalent strategies. Since each strategy is constructed by periodic real-time tasks, a strategy that is valid at the application level may not always be *feasible* at the system level. How to find feasible, schedulable, strategies from a a set of functionally equivalent strategies, given by the application,

*This research was supported in part by DARPA SDR DABT63-99-1-0022 and MARS DABT63-99-1-0004



$push$ strategy $pull$ strategy

Figure 1. Tasks in a *Leader/Follower* team

is one goal of this work.

The team size is not fixed, so, as robots enter or leave the team, the application must recompute the set of strategies that can be used. As the team size changes, the application may determine that the goal of the behavior must change as well, which, in turn, also changes the set of correct strategies that can be used. Figure 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 which is the square in the lower left of the map. Each time a robot joins the team and the set of possible strategies changes, the application must run the on-line scheduling algorithm to determine which strategies are feasible; otherwise, a system failure may occur. The control tasks that make up a strategy may be distributed among sites in a team, such as H_1, H_2 and L_2 in the *push* and *pull* models. A goal of our work is to determine the assignment of tasks to sites in order to optimize the coordinated behavior while minimizing the communication overhead and workload at each site, thereby improving overall schedulability.

As shown in the task graphs of *push* and *pull*, communication between tasks is needed to achieve coordination. In this paper, we assume that each robot is equipped with wireless broadcast communication. Because a shared multiple-access medium is used in the system, contention for the communication medium can occur at run time. We avoid this contention by scheduling the communication as well.

Assigning tasks with precedence relationships in a distributed environment is in general an NP-hard problem [17], and even some of the simplest scheduling problems are NP-hard in the strong sense [7]. Systematically derived heuristic allocation and scheduling algorithms are, therefore, proposed and evaluated in this paper.

The contributions of this paper are as follows. We develop an online 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 heuristics for allocating control tasks to distributed processing entities, which aim to improve the overall schedulability by minimizing communication costs and utilization of processors. We have performed extensive evaluations of the algorithms and also exercised 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.

The rest of the paper is structured 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. Results of evaluations from simulation are provided in Section 4. 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 site having an identical processor. In this paper, we use *site* and *processor* interchangeably. Robots in a team share a communication medium, which allows broadcast communication between robots. To prevent contention, communication is also prescheduled.

A *strategy*, which is specified at the application level, is denoted at the system level by an acyclic *Task Graph* (TG). To achieve a common goal for the team, a set of functionally equivalent strategies may also be supplied by applications.

In a TG, nodes represent tasks (T_i), directed edges between tasks represent *precedence* (e.g., *producer/consumer*) relationships. The amount of communication is denoted as a communication cost attached to the edges. All tasks in our model 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 producer and the consumer run with arbitrary periods, task executions may get out of phase, which results in large latencies in communication [21]. Harmonicity constraints can simplify the reading/writing logic and reduce those latencies [20]. Harmonic periods may also increase the feasible processor utilization bound [25]. To this end, we assume the period of the consumer is a multiple of that of the related producer.

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:

- within the LCM (Least Common Multiple) of task periods in that strategy, each instance of a task is scheduled at its schedule start time and the completion of this instance will not be later than its relative deadline;
- all constraints, such as *precedence*, are satisfied.

Based on the nature of the application, some tasks, e.g, sensor and motor systems, are required to run on designated sites, e.g., a specific robot platform. Other tasks, however, can be assigned to any site in a team. To find a feasible strategy, the system needs to:

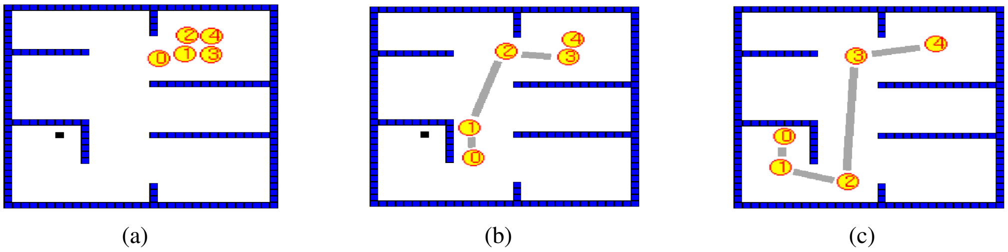


Figure 2. A sequence of active robots in a robot team

1. assign unallocated tasks to appropriate sites;
2. determine a schedule for all task instances.

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, where CCR_{ij} is defined as:

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

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_i	Site ID
u_i	Utilization of the site S_i
u_i^k	Utilization of the site S_i that T_k is on
$T_i \rightarrow T_j$	Precedence constraint between T_j and T_i
$CCR_{i,j}$	Communication Cost Ratio of $T_i \rightarrow T_j$

Table 1. Notation used in this paper

3.1 Allocation Algorithm

Optimal assignment of real-time tasks to distributed processors is an intractable problem. Two resource-bounded iterative heuristics are proposed in this section. Based on utilization of each processor and the amount of communication between tasks, the heuristics attempt to minimize workload of each processor, as well as the total communication. A dynamic utilization threshold is used at each step in both heuristics. The function of this threshold is to: 1) balance and minimize the workload of each processor; 2) avoid the violation of utilization bound for schedulability purpose.

3.1.1 Greedy Heuristic

This heuristic considers the amount of communication and computation involved for *each pair* of producer and consumer tasks. A decision is made as to whether these two tasks should be assigned to the same processor, thereby eliminating the communication cost. At each step, for all allocated tasks and related unallocated successors, the algorithm selects an unallocated task T_y that has the largest $CCR_{x,y}$, where T_x is located on site S_k and $T_x \rightarrow T_y$. The algorithm then attempts to allocate T_y to S_k, based on whether or not the utilization of S_k becomes larger than the threshold t. If the utilization is not larger than t, T_y is assigned to S_k and t needs not change; otherwise, the algorithm tries to find a site S_l that currently has the least utilization, and attempts to assign T_y to S_l. In this case, t may need to be updated. If a proper processor can be found, the algorithm continues with the next unallocated task that has the largest CCR among the remaining unallocated tasks, using the new threshold. If there is no task left to be assigned and the workload of every processor is less than 1, the algorithm is deemed successful; if the algorithm chooses a task to be allocated, but no site can be found (because any processor's utilization will be larger than 1 after loading this task), the algorithm fails. The pseudo-code for the Greedy allocation algorithm is shown in Table 3.

The basic idea of updating the threshold t is to use an increasing limit on utilization. Initially, t is the maximum value of the utilizations of all processors to which preallocated tasks have been assigned. At the times when the algorithm has to find a site S_l with the least utilization, so it can assign T_y to S_l, several conditions need to be considered. Suppose $t^{'}$ is the utilization if T_y is assigned to S_k, and $u_l^{'}$ is the utilization if T_y is assigned to S_l. First, if $t^{'} > 1$, in this case, if site $l \neq k$ and $u_l^{'} \leq 1$, then the task can be assigned to S_l and the threshold is updated to $max(t, u_l^{'})$; otherwise, no processor can be found for loading T_y since all utilizations will be larger than 1. Second, if $t^{'} \leq 1$, then T_y is assigned to S_l, but t is updated to t' since this is the expected lowest value since the last time. The pseudo-code for the function updating the threshold is shown in Table 4. The returned value is either the new threshold if it finds a location, or -1 if it does not.

3.1.2 Aggressive Heuristic

To address the motivation behind this heuristic, let us use a simple task graph depicted in Figure 3, for which WCETs and periods are given in Table 2. Consider the commu-

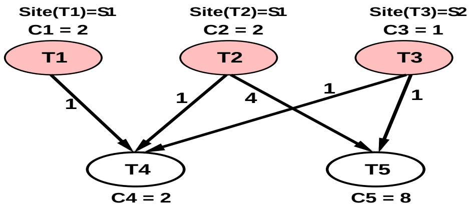


Figure 3. A simple task graph example

Task	T_1	T_2	T_3	T_4	T_5
WCET (C_i)	2	2	1	2	8
Period (P_i)	10	20	2	20	40

Table 2. Parameters for tasks in Figure 3

nications to task T_4. T_4 is assigned to site S_2 in the *Greedy* algorithm because $CCR_{3,4}$ is larger than $CCR_{1,4}$ and $CCR_{2,4}$. However, we notice the accumulated communication cost from S_1 is larger than that from S_2, i.e., $(CCR_{1,4} + CCR_{2,4}) > CCR_{3,4}$. So, intuitively, it is better to assign T_4 to S_1 instead of S_2.

The second heuristic we propose takes into account the total communication from the same site to see which unallocated task has the largest accumulated CCR, and then selects this task to be considered next. Since the utilization bound is still needed to be considered, once the task is selected, the assignment and threshold updates are the same as in the Greedy heuristic. To this end, line 5 in Table 3 needs to be changed to: Initialize $R = \{R^x_y | T_y \in N\}, R^x_y = \sum_i CCR_{i,y}, T_i \in F, T_i \to T_y$, Site$(T_i) = S_x$; and line 12 is changed to: $R = R \setminus \{R_y\} \cup \{R_z\}, \forall T_z \in N, T_y \to T_z$. Following the *Aggressive* algorithm, T_4 is first selected and assigned to S_1, and then T_5 is also assigned to S_1, since S_1 currently has the least utilization, at 0.4. This slightly more complicated algorithm is shown more effective with regards to higher schedulability in Section 4.

Greedy Allocation Algorithm
Input: a task graph $G = (E, V)$; P_i, C_i for each task T_i; communication costs; preallocated tasks with related sites; the number of sites m
Output: an assignment to all unallocated tasks such that utilization of each processor is less than 1
Variables:
F: the set of tasks that have been allocated;
N: the set of tasks that have not been allocated
R: the set of communication cost ratios[a]
U: array of utilizations (workload)
t: threshold of utilization
$CCR_{i,j}$: communication cost ratio of $\frac{communication\_cost(T_i \to T_j)}{C_i + C_j}$
Algorithm 3.1:
1. Initialize $U = \{u_i | i = 1, 2, \ldots, m\}$, such that 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 $t = max(u_i), u_i \in U$,
/* t is the threshold for workload control, initialize the threshold */;
3. If $(t > 1)$,**do**
4. exit without solution;
5. Initialize $R = \{CCR_{x,y} | T_x \in F, T_y \in N\}, T_x \to T_y$;
6. **While** (N is not empty) **do**
7. Find such task T_y that has the maximum value $CCR_{x,y}$ out of R;
8. Let $u^{'}_k = (u_k + \frac{C_y}{P_y})$;
/* $Site(T_x) = S_k$, calculate the new utilization, if T_y is allocated to site k */
9. **If** $((t = thresholdUpdate(u^{'}_k, k, T_y)) < 0)$, **do**;
10. exit without solution; /* cannot find an appropriate site */
11. Update set F, N such that $F = F \cup \{T_y\}, N = N \setminus \{T_y\}$;
12. Update set R, such that $R = R \setminus \{CCR_{x,y}\} \cup \{CCR_{y,z}\}$, $\forall T_x \in F, T_z \in N (T_x \to T_y) \wedge (T_y \to T_z)$.

[a]We use the same notation R to express different functionalities in two heuristics algorithms

Table 3. *Greedy* allocation algorithm

Function of Assignment and Threshold Update
$float$ **thresholdUpdate(float $t^{'}$, int k, Task T_y)**
/* t is the threshold; T_y and processor k is selected; $t^{'}$ is the new workload if assigning T_y to k */
1. **Case 1:** $t^{'} \leq t$, **do** /* $t^{'}$ is less than the threshold t */
2. Assign task T_y to processor k;
3. Update U with the new utilization $u_k = t^{'}$;
4. **Case 2:** $t^{'} > t$, **do** /* $t^{'}$ is larger than the threshold t */
5. Find the processor l that has least utilization $u_l = min(u_i), u_i \in U$, let $u^{'}_l = u_l + \frac{C_y}{P_y}$;
6. **Case 2.1:** $t^{'} > 1$, **do** /* processor k cannot load T_y */
7. **If** $(l \neq k) \wedge (u^{'}_l \leq 1)$, **do**
8. Allocate task T_y to processor l;
9. Update U with the new $u_l = u^{'}_l$;
10. $t = max(t, u^{'}_l)$;
11. **Else** return -1 ;
/* $(l = k) \vee (u^{'}_l > 1)$, cannot find a processor to load T_y */
12. **Case 2.2:** $t^{'} \leq 1$, **do** /* $u^{'}_l \leq t^{'} \leq 1$ */
13. Allocate task T_y to processor l;
14. Update U with the new $u_l = u^{'}_l$;
15. $t = t^{'}$;
16. **Return** t; /* new threshold */

Table 4. Assignment and threshold updates

3.2 Making Scheduling Decisions

After a successful assignment is found, a schedule is needed for each instance of tasks. Before we discuss the algorithm, 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 $E^1_i = 0$; and for the n^{th} instance of that task, $E^n_i = (n-1) \times P_i$, where $1 \leq n \leq N_i, N_i = L/P_i$. 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^1_i is set to $max(E^1_i, E^1_k + C_k)$, where $\forall k, T_k \in Predecessors(T_i)$. Since we will model communication as a task if the producer/consumer tasks are on different sites, and we have harmonicity constraints for all such pairs, initially, the lower bound of E^n_i is assigned to $(n-1) \times P_i + E^1_i$.

- **Communication task** If the producer and consumer are allocated on the same site, the communication cost is avoided; otherwise, communication needs to be scheduled. We model this as a communication task. For $T_i \rightarrow T_j$, a communication task T_{comm} has following features.

    1. $P_{comm} = P_j$. This is because each instance of T_j needs to process data sent from T_i only once during one period of T_j;
    2. $E^n_{Comm} = (n-1) \times P_{comm} + E^1_i$. This is a lower bound because the instance should begin execution at least after the completion of the first instance of T_i;
    3. $D^n_{comm} = n \times P_{comm} - C_j$. This is an upper bound since the communication should finish its execution no later than the latest start time of T_j.

Within the LCM, each instance of tasks is treated as an individual entity to be scheduled. We take into account the *deadline*, *laxity* and *earliest start time*, which characterize the most important properties for real-time tasks and precedence constraints, to actively direct the searching to find a feasible schedule. The potentially heuristic functions of H are: (1) Minimum 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) $H(T) = Min\_E + W \times Min\_L$; where W is the weight factor to adjust the effect of different temporal characteristics of tasks.

The search attempts to determine a feasible schedule for a set of tasks in the following way. 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 until a feasible schedule is derived. 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. Because Min_E considers the *precedence* constraints, it performs better than other simple heuristics in our experiments. The simulation studies also show that $(Min\_D + W \times Min\_E)$ has superior performance.

4 Simulated Results

To study the features of the proposed algorithms, we conducted several experiments to evaluate the allocation heuristics with regards to schedulability. How to use it for 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 the Communication Ratio used to assign communication costs. Experiments were conducted with CR values 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 $(1, maxP^O_j)$ 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: $maxP^O_j = mult\_factor \times max(maxP^I_i)$, 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 [20] to process output tasks; finally, we use the GCD technique for intermediate tasks to achieve harmonicity constraints. The idea of computing GCD is to do a backward period assignment: a task T_k gets period P_k from all its successors so that $P_k = GCD\{P_l | P_l \in succ(T_k)\}$. Because of precedence constraints, periods of output tasks cannot be considered separately from those of input tasks, so P^O_1, which is the least period of output tasks, is calculated upon the largest period of input tasks (P^I_m), $P^O_1 = \lfloor maxP^O_1 / P^I_m \rfloor \times P^I_m$. Other output tasks' periods are computed upon P^O_1 to achieve harmonicity.

- Parameter out_degree is used to set the precedence relationships in terms of data processed by multiple producers/consumers. For each task, except for output tasks, the out_degree is randomly chosen between 1 and 3. The total number of tasks in a task set is: $4 \times tasksetsize\_factor$, where $3 \leq tasksetsize\_factor \leq 8$, and all the results shown here are for task sets with four input and four output tasks, though we have conducted experiments with different numbers.

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 $\sum_{i=1}^{n}(C_i/P_i) \leq m$, where n is the number of tasks and m is the number of processors. For a given task set, if this condition is not held, at least one processor utilization will be larger than 1. The scheme used here is to remove the task sets that are definitely infeasible. 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.

Therefore, the performance of the algorithms and parameter settings are compared using the $SuccessRatio$ (SR):

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

N^{succ} is the total number of schedulable task sets found by the algorithm, and N is the total number of task sets tested. Here N is 100 for each simulation run, and for each result point in the graphs, $SR = (\sum_{i=1}^{10} SR_i)/10$, where $SR_i = N^{succ}/100$.

The tests involved a system with 2 to 12 processors connected by a multiple-access network. Resources other than CPUs and the communication network were not considered. Whereas we study the algorithm under various parameter settings, due to space limitation, here we only show some of the salient results and conclusions. Details that are not reported here can be found in [13].

4.1 Choosing a Scheduling Heuristic

In order to eliminate bias from the scheduling search heuristics, we first examine which heuristic function is suitable to evaluate the allocation algorithms in terms of Success Ratio (SR) of schedulability for a fixed task set size.

For both *Greedy* and *Aggressive*, we found Min_E is the best simple heuristic. This is because the *earliest start time* of each instance of a task encodes the basic *precedence* information. For integrated heuristics, $Min\_D + W * Min\_E$ has substantially better performance than other heuristics including Min_E. The reason should be clear: besides *precedence* constraints, another important factor, *deadline*, is also taken into account.

Since $Min\_D + W * Min\_E$ is a weighted combination of simple heuristics, we investigate its sensitivity to changes of weight (W) values. When $W = 0$, the heuristic becomes the simple heuristic Min_D, and does not perform well. When the weight increases from 0 to 4, or from 0 to 12 when $\#Proc = 2$, we see a significant performance increase for various $\#Proc$ values. The algorithm is robust with respect to heuristics, as performance is affected only slightly when the weight varies from 4 to 30 (or 12 to 30 if $\#Proc = 2$). So we will choose $W = 4$ for following experiments. Hereafter, we denote "*The Number of Processors*" as "*#Proc*", and "*The Number of Tasks Within a Set*" as "*#Task*".

4.2 Performance of the Allocation Algorithm

In this section, we evaluate the performance of the allocation algorithms, *Greedy* and *Aggressive*, compared with another method, *random* allocation. Figure 4 illustrates the results for task set size of 12, 20 and 32, respectively, when $CR = 0.1$. As shown in the graphs, for each instance, the performance of *Aggressive* is better than that of *Greedy*, which is in turn better than the random allocation. The gains come from the elimination of communication while maintaining minimal utilization for each processor. Since *Aggressive* takes into account the utilization bound at each assignment step, and tries to cluster as many tasks as possible, so as to eliminate the total communication cost, it is not surprising that it achieves better performance than *Greedy*, which considers only the individual communication cost for a given site.

The other observation is that the improvements in performance of *Greedy* or *Aggressive* with $CR = 0.4$ is larger than the improvement with $CR = 0.1$, especially when the task set size is large, say, no less than 20. Table 5 shows the difference in improvement of *Greedy* to the random; Table 6 is for the improvement of *Aggressive* to the random. Both are with $\#Task = 20$ and $\#Task = 32$, respectively. As we can see, in most cases, the improvements with $CR = 0.4$ are much larger than those with $CR = 0.1$ for either the *Greedy* or the *Aggressive* heuristic. The reason is that when $CR = 0.4$, the communication costs introduce more workload into the system, and hence increase the resource contention. 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 a resource-tight environment.

Finally, we found that as the number of processors increases, the improvements for both *Greedy* and *Aggressive* become less for a given task set. This observation shows that the tighter the resource constraint, the better our algorithms perform.

Processor	4	6	8	10	12
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

(a)$\#Task = 20$

Processor	4	6	8	10	12
CR = 0.1	4.7	13	7	2.9	0.3
CR = 0.4	4.5	12.1	11.4	12.3	13.1

(b) $\#Task = 32$

Table 5. Improvement of *Greedy* over random approach (Percentage)

4.3 Effect of Communication

Due to space limit, we omit details of the effect of communication costs that may be found in [13]. 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 intro-

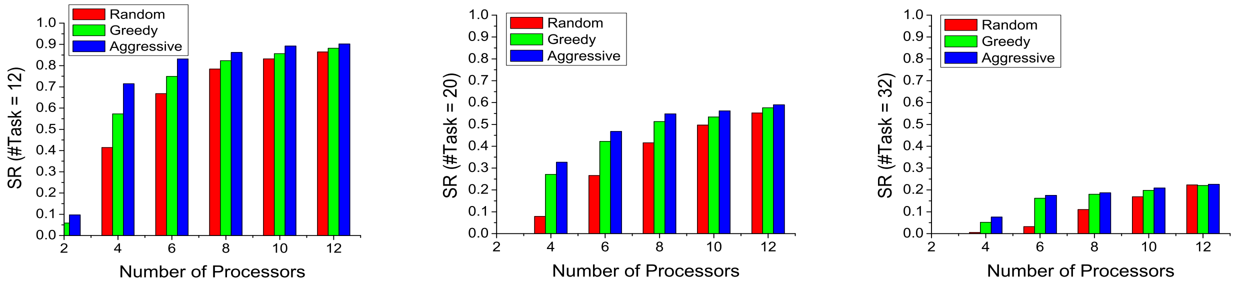


Figure 4. Effect of allocation algorithms (CR = 0.1)

Processor	4	6	8	10	12
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

$$\#Task = 20$$

Processor	4	6	8	10	12
CR = 0.1	7.1	14.3	7.7	4.0	0.3
CR = 0.4	6.4	12.5	11.8	13.3	13.6

$$\#Task = 32$$

Table 6. Improvement of *Aggressive* over random approach (Percentage)

duces additional precedence constraints that will affect the earliest start time of consumers. The communication costs also affect the overall system workload, therefore, lowering communication leads to improved schedulability.

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 7, the WCET of tasks are taken from an experimental implementation on a StrongARM 206MHz CPU; in Table 9, 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 and motor drivers by the application. Therefore, the periods of controller tasks are also designed to be 220 ms by the harmonic constraint. Though these figures are given based on tasks in Figure 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 strategies, composed of *push* and *pull* controllers, that the application can use for a team of three robots: $\{Pull, Pull\}$, $\{Pull, Push\}$, $\{Push, Pull\}$, and $\{Push, Push\}$. The task graphs for these four strategies are shown in Figures 5.

Task	$IR1(2)$	$Pos1(2)$	H_1	H_2	L_2	$M1(2)$
Push	20	120	35	25	5	20
Pull	20	120	25	25	18	20

Table 7. WCETs (ms) of tasks in Figure 1

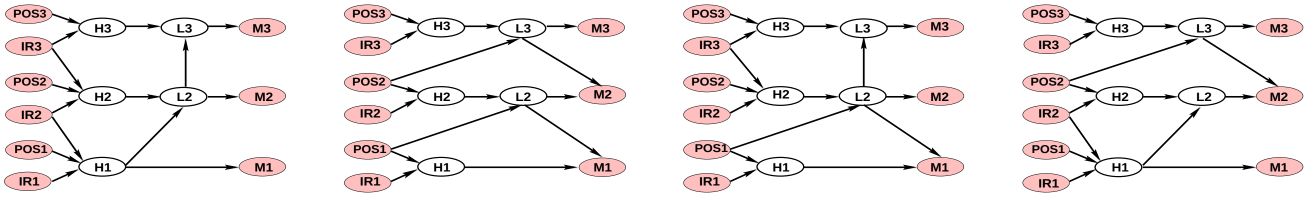
First, let us use the *Aggressive* heuristic to analyze the tasks' locations in each strategy. The details of the allocation steps are omitted here due to space limitations. In this example, since the accumulated communication cost is considered, the allocation is the same for all strategies: H_1 is assigned to S_1, H_2 to S_2, $H3$ to S_3, L_2 to S_2 and L_3 to S_3.

Next, the algorithm will see which strategies are schedulable using 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 8. 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, then the application will choose the $\{Pull, Push\}$ strategy, which has the maximal 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.

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

Table 8. The completion time for all strategies



(a) $\{Push, Push\}$ (b)$\{Pull, Pull\}$ (a) $\{Push, Pull\}$ (b) $\{Pull, Push\}$

Figure 5. Four possible strategies for a team of three robots using the *push* and *pull* controllers

Comm.	$IR_{1(2)} \to H_{1(2)}$	$Pos_{1(2)} \to H_{1(2)}$	$H_1 \to L_2$	$H_1 \to M_1$	$H_2 \to L_2$	$Pos_1 \to L_2$	$L_2 \to M_{2(1)}$
Push	0.02327	0.01236	2.979	2.979	2.979	0	2.979(0)
Pull	0.02327	0.01236	0	2.979	2.979	0.01236	2.979(2.979)

Table 9. Communication costs of Figure 1

6 Related Work

Numerous research results have demonstrated the complexity of design for real-time system, especially with respect to temporal constraints [9, 19, 20, 21, 22]. Also the schedulability analysis for distributed real-time systems attracted a lot of attention in recent years [12, 15, 16, 28]. For tasks with temporal constraints, researchers have focused on generating task attributes, e.g., period, deadline and phase. For example, Gerber *et al.* [9, 22] 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 constraintsfrom performance requirements are proposed by Seto *et al.* [23, 24]. 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.

For a set of *independent* periodic tasks, Liu and Layland [14] first developed the feasible workload condition for schedulability analysis under uniprocessor environments. Much later, Baruah *et al.* [5] presented necessary and sufficient conditions, namely, $U \leq n$ (n is the number of processors) based on *P-fairness* scheduling for multiprocessors. Also, the upper bounds of workload specified for the given schedules, e.g., EDF and RMA, are derived for homogeneous or heterogeneous multiprocessor environments [2, 4, 6, 10, 11, 26]. 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.

Peng *et al.* [17], Abdelzaher *et al.* [1] and Ramamritham [18] studied the task allocation and scheduling problem in a distributed environment. In their models, subtasks or modules of a task can have precedence and communication constraints. From this perspective, their work comes closest to ours. By using a branch-and-bound search algorithm [17], the optimal solution in the sense of minimizing maximum normalized task response time is found to the problem of allocating communicating periodic tasks to heterogeneous processing nodes. 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 [8], while the algorithm [3] used in their method is to find a preemptive schedule; 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 [1], a period-based method is proposed to the problem of load partitioning and assignment for large distributed real-time applications. Scalability is achieved by utilizing a recursive divide-and-conquer technique.[18] discussed a static algorithm for allocating and scheduling components of periodic tasks across sites in distributed systems. How to allocating replicates is a major issue counted 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 bound along each allocation step; 2) consider the *precedence* constraints as early 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 mobile robotic 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 utilizations 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

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

[2] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *IEEE real-time systems symposium*, pages 193–202, December 2001.

[3] 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 to release dates and precedence constraints. *Operations Research*, 31(2):381–386, March 1983.

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

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

[6] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In *IEEE real-time systems symposium*, pages 183–192, December 2001.

[7] M. R. Garey and D. S. Johnson. Strong np-completeness results: Motivation, examples, and implications. *ACM*, 25(3):499–508, July 1978.

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

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

[10] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems. Accepted for publication.*

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

[12] T. Kim, J. Lee, H. Shin, and N. Chang. 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.

[13] H. Li, J. Sweeney, K. Ramamritham, and R. Grupen. Pre-analyzed resource and time provisioning in distributed real-time systems: An application to mobile robotics. In *University of Massachusetts, Technical Report*, January 2003.

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

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

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

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

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

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

[20] M. Ryu and S. Hong. 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*, 1999.

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

[22] M. Saksena and S. Hong. 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*, pages 306–313, October 1996.

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

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

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

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

[27] J. Sweeney, T. Brunette, Y. Yang, and R. Grupen. Coordinated teams of reactive mobile platforms. In *Proceedings of the 2002 IEEE Conference on Robotics and Automation, Washington, D.C.*, May 2002.

[28] S. Wang and G. Farber. On the schedulability analysis for distributed real-time systems. In *Joint IFAC—IFIP WRTP'99 & ARTDB-99*, May 1999.