Pre-Analyzed Resource and Time Provisioning in Distributed Real-Time Systems: An Application to Mobile Robotics

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

Abstract

Our work is motivated by mobile robotic applications where a team of autonomous robots cooperate in achieving a goal, e.g., using sensor feeds to locate trapped humans in a building on fire. To attain the goal, (a) kinematic Line-Of-Sight (LOS) constraints must be enforced to build interference free communication paths, and (b) precedence, communication and location constrained tasks must be allocated and scheduled across the processing entities in a predictable and scalable fashion.

To maintain LOS constraints, symmetric task models are derived and coordinated. To improve the schedulability of these tasks and to make the predictable scalability analysis possible in the presence of precedence, communication and location constraints, both task dependencies and workloads are examined with the objective of minimizing communication costs and workload of each processor.

Extensive experimental results using simulated scenarios show that the allocation and scheduling techniques incorporated in our approach to addressing these issues are effective in improving the performance of the systems, especially in resource limited environments. A real example application drawn from mobile robots is also used to demonstrate the significant benefits that accrue from the proposed method.

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

I. INTRODUCTION

One promising application for a team of mobile robots is searching a burning building for trapped people. Assume that each robot has a position sensor, an obstacle detection sensor, and wireless communication capable of point-to-point communication with other robots within a fi xed transmission range. Given that the transmission range imposes a constraint on the movements of the robots, one approach to performing the search task is for the robots to form a serial kinematic chain, where each member in the chain is always within communication distance of its neighbors. In the example of exploring a burning building, a mobile robot that is inside the building will probably not be able to communicate directly with an operator outside, either because its transmitter is not powerful enough, or because the walls of the building interfere with communications. By forming a multi-robot chain, the leader of the search can communicate with the operator by relaying data through the chain. Thus, in order for the chain to function, each robot must be able to communicate with its neighbors. In other words, two robots must be within communication range and there must be an interference-free path between the robots forming the team for communications to occur. This is known as the Line-Of-Sight (LOS) constraint.

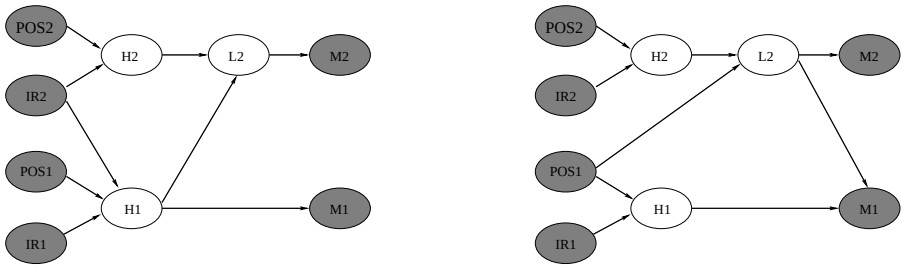
(a) (b) (c)

Fig. 1. A leader/follower coordination primitive in action. The leader robot is moving from right to left

To perform this search task, each robot must address multiple, concurrent objectives: either searching and maintaining LOS with a follower, or maintaining LOS with both a leader and a follower. A pairwise coordination strategy for multi-objective, concurrent controllers in a distributed robot team is presented in [27]. Figure 1 shows three frames of a search task with LOS constraints between the leader and the follower, where the leader is the bottom robot. The robots are equipped with IR sensors used to detect and map obstacles in the environment. LOS is maintained by computing the physical area of the environment which is free of occluding obstacles and within transmission range, referred to as the "LOS region."

The pairwise LOS preserving controllers have two symmetric modes, denoted *push* and *pull*. By symmetric, we mean that the task model of the system has a symmetric structure that can be easily generalized to accommodate additional robots. The push and pull controllers differ in the way in which the LOS region is computed and communicated between the pair. The task models for push and pull for two robots are shown in Figure 2. 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, which takes desired heading and speed commands and turns them into motor commands, denoted M_i. All three of these tasks are specifi c to the hardware of each robot, so they are all preallocated to run locally on each robot, with the follower and leader denoted $i = 1$ and $i = 2$, respectively (the numbers attached in gray ellipses show the site where the tasks are located). In the push confi guration, the follower computes the LOS region in task H_1 (standing for the abbreviation of the path planner), and passes it to task L_2, which computes a new movement vector of the leader that maximizes the search while keeping the leader within the specifi ed LOS region. In the pull confi guration, the leader robot searches, and, concurrently, computes the LOS region in task L_2. This LOS region is based on the leader's local knowledge of the environment, gained through its IR sensor task IR_2, and the follower's position task POS_1. The qualitative difference between the two confi gurations is that with the pull controller, the leader is able to "pull" the follower along while it performs the search, by specifying the LOS region to the follower. In contrast, using the push confi guration, the leader's search task is limited in a way such that the leader must always be in the LOS region assigned by the follower. Thus the follower is in effect "pushing" the leader along.

The three control tasks H_1, H_2, and L_2 may reside on a single robot, or be distributed between the pair, if necessary, to optimize processor utilization or communication costs. The functionality of the team is not affected by changing the allocations of the control tasks. The sensor and motor tasks IR_i, POS_i, and M_i are designed to be perform periodic updates. The control tasks are hence forced to execute periodically in order to consume the new sensor data and give new motor commands. Consequently, the periods of the control tasks must be assigned



(a) Task model for push strategy (b) Task model for pull strategy

Fig. 2. Tasks in Leader/Follower Robot Team with LOS Constraint

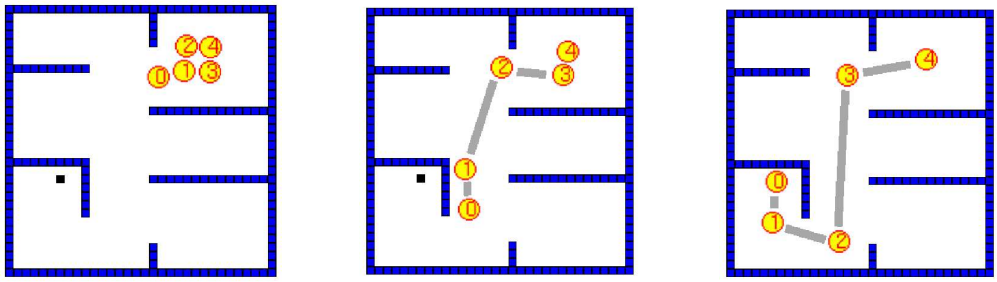
based on the periods of the sensor and motor tasks.

Intertask communication can occur through shared memory if the tasks execute on the same robot, or they must use a shared communication channel if the tasks are distributed across different robots, which in this case is assumed to be a wireless medium. In any case where tasks on different robots utilize the shared channel, a non-negligible communication cost is incurred, that the scheduling and allocation must take into account. In the push and pull coordinated control example, the locations of the control tasks H_i and L_i determine the total communication costs. For example, in the push configuration, if H_2 and L_2 reside on the leader robot, and H_1 on the follower, then the communications are $IR_2 \rightarrow H_1$ and $H_1 \rightarrow L_2$. If L_2 is moved to the follower, then the cost $H_1 \rightarrow L_2$ is avoided, but new costs $H_2 \rightarrow L_2$ and $L_2 \rightarrow M_2$ are introduced.

Which allocation strategy is better depends on the communication costs, the precedence relationships, the utilization of each processor, and finally, the scheduling scheme. So, in addition to periodicity constraints for tasks, several constraints caused by task dependencies must also be considered, including: locality, communication, and precedence constraints. All these constraints are related to each other. Locality and communication constraints affect the allocation of unallocated tasks which, in turn, affects the workload of processors. Precedence constraints affect the earliest start time of information consuming tasks, if they are expected to consume up-to-date data. Besides costs, communication relationships introduce two precedence constraints: *producer* $\rightarrow$ *communication_task* $\rightarrow$ *consumer*. Even though communication costs can be ignored if two communicating tasks are on the same node, their precedence relationship cannot be neglected. Obviously all these constraints affect the schedulability of the tasks in a given system.

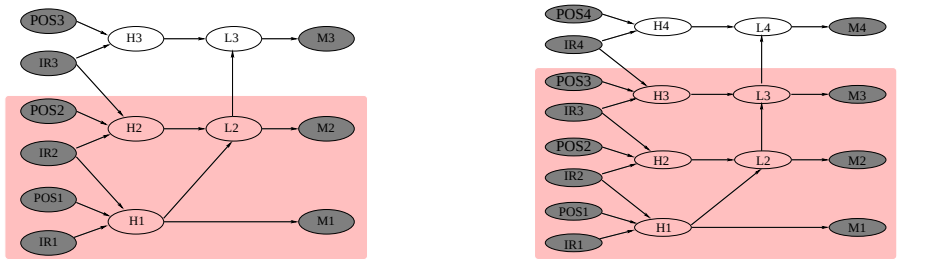
In teams of mobile robots, the other important but difficult issue is predictable scalability. Figure 3 shows a sequence from a simulation with five robots, where robot 0 is the leader searching for the goal which is the square in the lower left of the map. The gray lines between robots represent the LOS constraint between pairs of robots. Figure 4 depicts task graphs for such teams with three and four active robots using the push strategy. The interesting observation is that the shadowed area in Figure 4(b), where $n = 4$, is the same as the task set with $n - 1 = 3$ (Figure 4(a)), except for the communication to and from the robot 4. Similarly, the subset shadowed in Figure 4(a) for 3 robots is the same as the task set for 2 robots (Figure 2(a)) with the similar exception, and so on for structures with $n > 2$ members. The question arises: what does this systematic model imply if the team of n robots were to maintain the LOS property? what's the largest n that can allow us to make feasible allocation and scheduling decisions, in other words, how scalable is our coordinated control configuration?

Development of techniques that makes autonomous robot teams scalable and schedulable is the major contri-



(a) (b) (c)

Fig. 3. A sequence of active robots in a robot team



(a) Push Tasks in a Three Robot Team (b) Push Tasks in a Four Robot Team

Fig. 4. Tasks in Three/Four Robot Teams with LOS Constraint

bution of this paper. The algorithms we propose allocate unallocated tasks to processing nodes, and schedule all tasks such that the tasks meet their deadlines. The challenges of solving such problems in the symmetric communication model include scaling the assignment strategy with an increase in the number of robots involved, and keeping the schedulability guaranteed.

Assigning tasks with precedence relationships in a distributed environment is in general an NP-hard problem [16]. And even some of the simplest scheduling problems are NP-hard in the strong sense [7]. Hence, it is not possible to determine optimal allocation and schedules effi ciently. When an additional robot is added, an exponential increase in computational time may be required to obtain an optimal result, especially when the team size is large. A systematically derived heuristic allocation and scheduling strategy is therefore proposed and evaluated in this paper. Since some of the heuristics built into the task scheduling algorithm of [19] were shown to be effective, we capitalize on them in developing the approach used in this paper. The results from our experiments show that our allocation and scheduling strategies indeed improve the schedulability compared to an approach (a) that does not explicitly deal with the task and system constraints or, (b) where the allocation decisions, especially in resource limited environments, are not factored into the schedulability analysis.

The rest of the paper is structured as follows: Section II discusses related work. In Section III, task characteristics and constraints considered by the algorithm are formulated. An overview of our solution is also given in this section. The details of the algorithm are provided in Section IV. Results of evaluation of the algorithm using synthetic task sets is provided in Section V. In Section VI, the application of our approach to a real-world example of a distributed robotic system is analyzed and Section VII concludes the paper by summarizing the important characteristics of the algorithm and discussing future work.

II. Related Work

Numerous research results have demonstrated the complexity of real-time design, especially with respect to temporal constraints [9],[20],[22],[18],[21]. The schedulability analysis for distributed real-time systems has also received a lot of attention in recent years [14],[15],[12],[28]. For tasks with temporal constraints, researchers have focused on generating task attributes (e.g., period, deadline and phase) with the objective of minimizing the utilization and/or maximizing system schedulability while satisfying all temporal constraints. For example, Gerber, Hong and Saksena [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 constraints, e.g., maximum sensor-to-actuator latency and minimum sampling periods from performance requirements are proposed by Seto, *et al*. [24],[23]. When end-to-end temporal 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 temporal characteristics and the allocation of real-time tasks. A more comprehensive approach that takes into consideration task temporal characteristics and allocations, in conjunction with schedulability analysis, is required.

For a set of independent periodic tasks, Liu and Land [13] first developed the feasible workload condition for schedulability analysis, e.g., $U = m(2^{1/m} - 1)$ for Rate Monotonic Algorithm (RMA) and $U = 1$ for Earliest Deadline First (EDF) 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 some given schedules, e.g., EDF and RMA, are derived and analyzed for homogeneous and heterogeneous multiprocessor environments [26],[11],[10], [2],[6], [4]. All these techniques are for preemptive tasks and task or job migrations are assumed to be permitted without any penalty. However, if precedence and communication constraints exist among tasks as in our task systems, these results cannot be used directly. Instead, an approach to task assignment that can take into account the relationship of workload and schedulability must be explored for distributed real-time systems.

Abdelzaher and Shin [1], Ramamritham [17] and Peng *et al*. [16] studied the problem where subtasks or modules of a task can have precedence and communication constraints in a distributed environment. Both allocation and scheduling decisions are considered in their approaches. In this respect, their work comes closest to ours. But there are many differences. In their task models, subtasks or modules within a task that are to be scheduled share the same period, while our algorithm handles tasks with different periods. In [1], Abdelzaher and Shin first generate task and processor clustering recommendations based on heuristic analysis of communication capacities and task periods, respectively. Task clusters are then mapped to processor clusters. Workload is considered in the mapping stage. However, schedulability is not further analyzed in their approach. In [17], workload of processors is not taken into consideration when the allocation algorithm make its decision on whether two communicating tasks are to be assigned to the same site. A branch-and-bound search algorithm for assigning and scheduling real-time tasks with the objective of minimizing maximum normalized task response time is presented in [16]. Even though the heuristic employed at nonterminal search vertices guides the algorithm efficiently towards an optimal solution, the algorithm cannot be simply applied and extended to our situation. Firstly, in their task

system model, tasks are the allocation entities and modules are the smallest schedulable objects. Hence, the search space will explode if allocation and scheduling objects are the same smallest one as in our environment. Secondly, the algorithm used in [3] for scheduling and calculating the lower bound of the non-leaf nodes finds a preemptive schedule on a single machine. But we consider a non-preemptive schedule which is NP-hard in the strong sense, even without precedence constraints [8]. Finally, if producer and consumer tasks have different periods, the precedence constraints are predetermined among specific instances of tasks in their case. However, since a task may consume data from more than one consumer, and one data may be consumed by more than one consumer in our case, the predetermined approach is inapplicable. In fact, our algorithm gives more flexibility to let the scheduler make the decision: which instance of the consumer will consume data from which instance of the producer depends on the scheduling that satisfies the precedence constraints.

III. Problem Formulation and Overview of Our Solution

A. System Model

We model a distributed system as a set of m sites $S = \{S^j | j = 1, 2, ..., m\}$, each site having one processor. So, in this paper, we use $site$ and $processor$ interchangeably. Also, in the interest of space, we study only the case where all processors are identical; heterogeneous system is discussed in the conclusion section as future work. The sites are connected by a shared communication medium. To prevent contention occurring for the communication medium at run time, communications should be prescheduled.

B. Task Characteristics

A directed acyclic task graph (TG) is used to represent a set of n tasks $T = \{T_i | i = 1, 2, ..., n\}$, where, the nodes in the graph represents tasks, and directed edges represent precedence (e.g., producer/consumer) relationships. The properties of the real-time tasks in such a task model are:

- the *period* P_i of task T_i. The period of a task defines the inter-release times of instances of the task. One instance of the task should be executed every period. Given the period of task T_i, the n^{th} instance of T_i will be released at time $t_n = (n-1) \times P_i$.
- the relative *deadline* D_i^n of the n^{th} instance of task T_i. The deadline specifies the time before which each task instance must complete its execution. In this paper, we assume that the deadline is equal to the period for the given tasks in the TG. Therefore, the n^{th} instance of T_i must complete execution before the relative deadline: $D_i^n = n \times P_i$.
- the *computation time* C_i of task T_i. The computation time represents the Worst Case Execution Time (WCET) of each task instance.

In addition to above task characteristics, additional constraints may be specified by system designers, such as:

- *precedence* relationship. Precedence relationships constrain the execution order of the tasks and the production and consumption relationships on the data flow. If $T_i \rightarrow T_j$, and the period of T_i is equal to that of T_j, let E_i^n be the earliest start time (the time all required resources are available) of the n^{th} instance of task T_i,

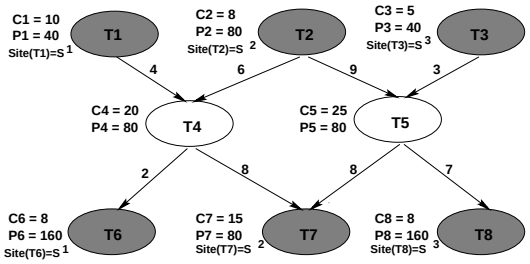


Fig. 5. Sample Task Graph with Locality and Communication Constraints

the constraint can be defined as :

$$\forall T_i, T_j (T_i \rightarrow T_j \wedge P_i = P_j) \Rightarrow (E_i^n + C_i \leq E_j^n)$$

If $T_i \rightarrow T_j$, and the period of T_i is different from T_j, reading/writing logic may be complicated and depends on the LCM (Least Common Multiple) of these two tasks. For instance, if the period of T_i is 3 and the period of T_j is 5, there are five instances of T_i and three instances of T_j to be executed within the LCM (15). Which data produced by T_i will be consumed by T_j depends on the scheduling scheme.

- *locality* constraint. Based on the nature of the environment, some tasks, *e.g.*, sensor and motor systems, are required to be executed on designated sites, *e.g.*, specific robot platform. The locality constraint for task T_i is defined as:

$$Site(T_i) = S^k$$

- *communication* constraints. Communication between tasks that are on different sites requires a communication medium and time to send/receive the message. Messages sent between tasks must be scheduled with the communication medium as a resource requirement. If we model the communication between each pair of communicating tasks as a special task, this task must satisfy the precedence constraints with the two communicating tasks separately. Let T_{comm} be the communication task, the precedence communication constraint is defined as:

$$\forall T_i, T_j (T_i \rightarrow T_j) \wedge (Site(T_i) \neq Site(T_j))$$

$$\Rightarrow \exists T_{comm} (Ti \rightarrow T_{comm} \rightarrow T_j)$$

- *harmonicity* constraints. If we model the communicating tasks as producers and consumers from the data production/consumption perspective, and if the producer and the consumer run with arbitrary periods, task executions may get out of phase resulting in large latencies in communications [21]. Harmonicity constraints are used to simplify the reading/writing logic and reduce the communication latencies [20]. Also harmonic periods increase the feasible processor utilization bound [25]. To this end, the period of the consumer is often made to be a multiple of the period of the producer:

$$\forall T_i, T_j (T_i \rightarrow T_j) \Rightarrow \exists n > 0 (P_j = n \cdot P_i)$$

Figure 5 gives an example of such a task graph. (We use this task graph for illustration since it is more general than the tasks of Figure 2.)

C. Our Goal and Overview of Our Solution

Given a distributed symmetric real-time system with task characteristics and constraints described in Section III-B, our goal is to: (a) allocate sites to tasks, and (b) determine schedule times for all task instances that make scalability through schedulability possible.

To solve the problem, we present an off-line strategy that consists of four steps.

1. The first step of the algorithm assigns unallocated tasks to sites, which is described in detail in Section IV-A. Two important issues that affect schedulability are considered in this part: 1) communication cost, and 2) the workload of each processor. If we want to increase schedulability by eliminating communication costs as much as possible, then the more communicating tasks placed on the same site, the better. But, this will increase the workload of a site, which in turn might decrease schedulability. Heuristics that take into account the trade-off between the communication cost and processor workload are proposed. The basic idea is to cluster the tasks that incur a high communication cost on the same site, while minimizing the utilization of each processor. This decision is based upon the Communication Cost Ratio (CCR) of a pair of communicating tasks $T_i \rightarrow T_j$:

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

   A dynamic workload threshold is used, at each allocation step, to balance and minimize the load.
2. The second step, discussed in Section IV-B, is to construct communication tasks. The algorithm uses communication tasks to model the communication cost and channel contention that occurs if communicating tasks happen to be on different sites. The execution time of the communication task is the communication cost; the period and the deadline of the task is computed based on the period and the latest start time of the consumer, respectively.
3. The third step is to construct a comprehensive graph containing all instances of all tasks including communication tasks that will execute within the LCM of task periods, and preprocess precedence relations of tasks by setting up the relative earliest start time of consumers. These details are in Section IV-C
4. Finally, a search is used to find a *feasible* schedule. The algorithm described in Section IV-D takes into account various task characteristics, in particular, deadline, earliest start time and laxity.

If the tasks are schedulable, then the original system-level constraints are satisfied; otherwise the task attributes, such as periods or deadlines, are adjusted until a schedulable task set is found, else the specifications of the application are reevaluated by the system designers.

IV. Details of the Algorithm

We now give the details of the algorithm. Notation used in the algorithm is explained in Table I.

A. Allocating Tasks

The optimal assignment of tasks to distributed processors is an intractable problem. We propose two resource-bounded heuristics to solve the allocation problem. Based on the communication costs and utilization of proces-

Notation	Meaning
T_i	Task id(ID)
P_i	Period of task T_i
C_i	Worst case execution time of task T_i
E_i^n	Earliest start time of n^{th} instance of T_i
D_i^n	Deadline of 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 I
NOTATION RELATING TO TASK AND SYSTEM CHARACTERISTICS

sors, the two heuristics attempt to minimize communication cost and workload of each processor. The difference between the algorithms lies in whether or not to cluster *a set* of communicating costs when selecting the next task to be allocated.

A.1 Greedy Heuristic

This heuristic considers the amount of communication and computation involved for a pair of communicating tasks. A decision is made as to whether the two tasks should be assigned to the same processor, thereby eliminating the communication cost. At each step, the algorithm chooses an unallocated task T_j that has the largest Communication Cost Ratio, $CCR_{i,j} = communication\_cost(T_i \rightarrow T_j)/(C_i + C_j)$, among all such ratios, where T_i has been allocated to processor k and T_j is to be assigned. The algorithm attempts to allocate T_j to the same processor k as T_i. The decision is made upon whether or not the utilization of k becomes larger than a threshold t if T_j is allocated to k. If the utilization u is not larger than t, it allocates task T_j to k and does not need to update t; otherwise, the algorithm tries to find a processor l that currently has the least utilization and attempts to allocate T_j to l. In this case, t may need to be updated. If a proper processor can be finally found, the algorithm continues with the next unallocated task that has the largest CCR of all remaining unallocated tasks.

The basic idea of updating the threshold t is to use increasing limits on utilization. Initially, t is the maximum value of the utilizations of all processors to which preallocated tasks have been assigned. At each time, if needed, t is updated according to where T_j is assigned. Consider the selected task T_j, processor k, and l that have been discussed before, the pseudo-code for the function updating t is shown in Table VIII. The returned value is either the new threshold if it finds a location, or -1 if it does not.

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 each processor's utilization will be larger than 1 if allocated this task), the algorithm is deemed to have failed in its allocation step.

The pseudo-code for the greedy allocation algorithm is shown in Table IX. Next, we will see how this algorithm works for the task set in Figure 5, i.e., how it derives allocations for T_4 and T_5. The computation times, periods, and locality constraints are given in Table II. The resulting site assignments are shown in Table II in **boldface**.

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
WCET (C_i)	10	8	5	20	25	8	15	8
Period (P_i)	40	80	40	80	80	160	80	160
Site	S^1	S^2	S^3	$\mathbf{S^2}$	$\mathbf{S^3}$	S^1	S^2	S^3

TABLE II
PARAMETERS AND ASSIGNED SITES FOR TASKS OF FIGURE 5

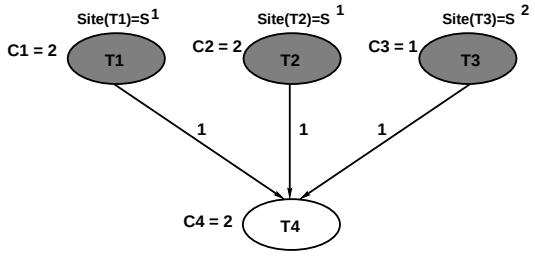


Fig. 6. A simple example of communicating tasks

Let us assume that there are 3 sites in the system. Initially the utilizations of sites are: $u_1 = C_1/P_1 + C_6/P_6 = 0.3, u_2 = C_2/P_2 + C_7/P_7 = 0.2875$, and $u_3 = C_3/P_3 + C_8/P_8 = 0.175$, so the threshold $t = 0.3$. The maximum communication cost ratio is $CCR_{2,5} = 9/(8+25) = 0.27$. If T_5 is allocated to site 2, the utilization of site 2 will be $u_2 = 0.2875 + 0.27 = 0.5575 > 0.3$, therefore the algorithm finds that site 3 has the least utilization, and because $u_3 + 0.27 = 0.445 < 1$, it allocates T_5 to site 3. Because $0.5575 \leq 1$, the algorithm updates the threshold $t = 0.5575$, and updates $u_3 = 0.445$. After allocating task T_5 to site 3, the algorithm begins a new iteration and considers the current maximum communication cost ratio $CCR_{2,4} = 6/(8+20) = 0.214$ and task T_4. If it allocates T_4 to site 2, $u_2 = 0.2875 + 0.214 = 0.5015 < t = 0.5575$. So the algorithm assigns T_4 to site 2 and updates $u_2 = 0.5015$. Since there is not task left and all utilizations are less than 1, the algorithm stops with a successful assignment. The final utilizations are : $u_1 = 0.3, u_2 = 0.5015, u_3 = 0.445$.

A.2 Aggressive Heuristic

Consider a simple situation with three located tasks and one unallocated task shown in Figure 6. The communication cost ratios are $CCR_{1,4} = \frac{1}{4}, CCR_{2,4} = \frac{1}{4}$ and $CCR_{3,4} = \frac{1}{3}$. Therefore, if T_4 is selected to be assigned, and we assume the utilization is no larger than the threshold, it will be allocated to site S^2 by the greedy heuristic. However, intuitively, we notice that the accumulated communication cost from S^1 is greater than the cost from S^2 even though the individual cost from S^1 is less than that from S^2. So we might need to consider a more sophisticated heuristic that takes into account the sum of all communication cost ratios from the same site. That is, at each step, when selecting an unallocated task T_j, we cluster the communications from the same source sites, and calculate the sum of CCRs, e,g., $\sum_i CCR_{i,j}$, where $T_i \rightarrow T_j$, and T_is are placed on the same site. Then we compare the values and select the one with the largest value. For instance in Figure 6, $CCR_{1,4} + CCR_{2,4} = \frac{1}{2} > CCR_{3,4}$, hence, T_4 should be considered to be assigned to site S^1 instead of site S^2.

The second heuristic we propose is to take into account the total communication from the same site to see

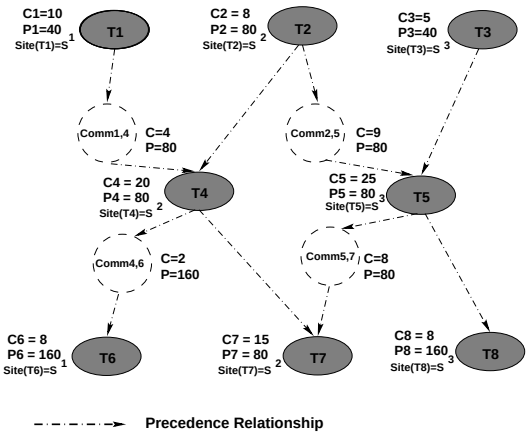


Fig. 7. Extended Task Graph Containing Communication tasks

which unallocated task has the largest accumulated communication cost ratio, and then we select this task to be assigned next. Once the task is selected, the assignment and threshold update strategy is the same as in the greedy algorithm. To this end, only line 5 in Table IX needs to be changed to: Initialize $R = \{R_y^x | T_y \in N\}, R_y^x = \sum_i CCR_{i,y}, T_i \in F, T_i \rightarrow T_y$, Site$(T_i) = S^x$; and line 12: $R = R \setminus \{R_y\} \cup \{R_z\}, \forall T_z \in N, T_y \rightarrow T_z$.

B. Construction of Communication Tasks

As mentioned earlier, if two tasks are placed on the same site, the communication cost between them is avoided and no communication task needs to be constructed. Figure 7 shows the extended task graph that includes communication tasks constructed from Figure 5 after allocating sites for T_4 and T_5. Consider the constructed communication task T_{comm} in $T_i \rightarrow T_{comm} \rightarrow T_j$, it has following properties:

1. Its period P_{comm} is equal to the consumer's. This is because each instance of the consumer task only needs to process data sent from the producer once during a period;
2. The earliest start time of the n^{th} instance is: $E_{Comm}^n = (n-1) \times P_{comm} + E_i^1$, where E_i^1 is the earliest start time of the first instance of T_i. This is a lower bound on its start time because it should begin execution at least after the completion of the first instance of the producer;
3. The deadline of the n^{th} instance is: $D_{comm}^n = n \times P_{comm} - C_j$. This is an upper bound because it should finish its execution no later than the latest start time of the consumer.

C. Constructing a Comprehensive Graph and Preprocessing Precedence Constraints

Given the extended task graph, the first step of this part is to construct a *comprehensive graph* that is composed of $N_i = \frac{L}{P_i}$ instances for each task T_i including communication tasks, where L is the Least Common Multiple (LCM) of the periods of all tasks. Because for a periodic task T_i, the n^{th} instance should be completed between $(P_i \times (n-1))$ and $(P_i \times n)$, the algorithm attempts to build a feasible schedule for all instances of tasks within L. And since all tasks in the comprehensive graph have only precedence constraints, it turns out that by applying certain preprocessing to the earliest start time or deadlines of related tasks we can ensure that the schedule algorithm automatically enforces the precedence relationships. Two cases are to be considered in the second step to preprocess the earliest start time: task with or without predecessors.

1. If task T_i has no predecessors, the first instance is ready to execute at time 0 with $E_i^1 = 0$, and for the n^{th} instance of the task, the earliest start time is $E_i^n = (n-1) \times P_i,\ 1 \leq n \leq N_i$.
2. If task T_i has predecessors, its first instance becomes enabled only when all its predecessors have completed execution. In order to achieve such a condition, the tasks in the original task graph are topologically ordered (from inputs to outputs) and when a task T_i is processed, the lower bound of the earliest start time of its first instance is set to $E_i^1 = max(E_i^1, E_k^1 + C_k)$, where $\forall k, T_k \in Predecessors(T_i)$. Since we have modeled communication as a task if the producer and consumer are on different sites, and we have harmonicity constraints for all producer and consumer pairs, initially the lower bound of the earliest start time of the n^{th} instance of task T_i can be assigned to $(n-1) \times P_i + E_i^1$.

Except for the communication tasks, the deadline of the n^{th} instance of T_i is defined as $D_i^n = n \times P_i, 1 \leq n \leq N_i$. The deadline of each instance of the communication task is given in Section IV-B.

D. Making Scheduling Decisions

Finally, we use search technique to find a feasible schedule. Every instance to be scheduled is considered as an individual task. Searching for the feasible schedule is based on the algorithm of [19] that finds a schedule for dynamically arriving tasks. However, here we apply this algorithm to a static environment, and therefore the earliest start time of each task is dictated by the precedence relationships instead of by arrival times. Given a set of tasks with earliest start times, deadlines, worst case execution times, and resource requirements, scheduling such a set of tasks to find a feasible schedule is to determine, T_Sch, the schedule time for each task. This is the time at which the task can begin execution, in other words, the time when resources required by the task are available, and all of its related precedent tasks have completed execution before their deadlines.

The search process is structured as a search tree starting with an empty partial schedule as the root. By moving to one of the vertices at the next level, the search tries to extend the schedule with one more task until a feasible schedule is derived. It might take an exhaustive search to find a feasible schedule, an intractable computation in the worst case. Therefore we take a heuristic approach.

To actively direct the searching to a plausible path, the heuristic function used for selecting the next task should synthesize various characteristics affecting real-time decisions. 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. While extending the partial schedule, the algorithm determines whether or not the current partial schedule is *strongly-feasible* [19]. A partial schedule is said to be strongly-feasible if all the extensions of the schedule with any one of the remaining tasks are also feasible. Given a heuristic function, the algorithm starts with an empty partial schedule and each step of the search involves:

- determining if the current partial schedule is strongly-feasible, if so:
- select the task T with the least heuristic value from the remaining tasks and extend the current partial schedule with this task.

Potential heuristic functions H include:

1. Minimum deadline first: $H(T) = Min\_D$;

2. Minimum earliest-start-time first: $H(T) = Min\_E$;
   (Earliest-start-time is precedence driven as defined in Section IV-B and IV-C for communication tasks and given tasks respectively)
3. Minimum laxity first: $H(T) = Min\_L = min(D_i - (E_i + C_i))$;
4. $H(T) = Min\_D + W * Min\_E$;
5. $H(T) = Min\_D + W * Min\_L$;
6. $H(T) = Min\_E + W * Min\_L$;

The first three are simple heuristics and the last three are integrated heuristics. W is the weight factor that is used to adjust the effect of different temporal characteristics of tasks. Because Min_E takes into account the precedence constraints, it performs better than other simple heuristics in our experiments. The simulation studies also show that $(Min\_D + W * Min\_E)$ has superior performance.

V. Evaluation of the Algorithm

We have implemented the above algorithm in C++ and have tested it with many synthetic task sets with constraints discussed before (Subsequently, we discuss the application of our algorithm to a robotic scenario). All the periodic tasks in our task set have precedence constraints that are represented as a directed acyclic graph and have 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 attached to a directed edge in the precedence graph 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 facilitate the system design, 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). The period of an input/output task can be a value within the range. In the experiments, the range for input task T_i is set to: $minP_i^I = Lower \times C_i$ and $maxP_i^I = Upper \times C_i$, where $Lower = 1.1$ and $Upper = 4.0$. Also 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 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 address harmonicity constraints, the algorithm first processes the input tasks to make their periods harmonic, and then processes the output tasks. We tailor the techniques from [20] to assign harmonic periods to output tasks and use the GCD technique for intermediate tasks to achieve harmonicity constraints. Whereas, in their approach, they choose a random period P_1^O such that $\lceil maxP_1^O/2 \rceil \leq P_1^O \leq maxP_1^O$ for the first output task that has the minimum $maxP$, in our case, periods of output tasks cannot be considered separately from those of input tasks, and this is achieved by calculating P_1^O based on the maximum period of input tasks (P_m^I), $P_1^O = \lfloor maxP_1^O/P_m^I \rfloor \cdot P_m^I$, other output tasks' periods are calculated upon P_1^O to achieve

harmonicity. The idea of computing the GCD is to do a backward period assignment: a task T_k gets period P_k from all its successors so that $P_k = GCD\{P_l | P_l \in succ(T_k)\}$.

- Since a consumer can consume data generated from different producers and data generated by one producer can be consumed by more than one consumer, the parameter out_degree was used to set the precedence relationships. For each task except output tasks, the out_degree is randomly chosen between 1 and 3.
- Even though we have conducted experiments with different numbers of input and output tasks, all the results shown here are for task sets with four input and four output tasks. The total number of tasks in a task set is: $4 \times tasksetsize\_factor$, where $3 \leq tasksetsize\_factor \leq 8$.

All the simulation results shown in this section are obtained from the average of 10 simulation runs. For each run, we generate 100 test sets with 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 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. As we know, determining the feasibility of a given task set is a computationally intractable problem [17], therefore if a heuristic algorithm does not succeed in finding a feasible schedule, it could be due to the infeasiblility of the task set. If one heuristic scheme is able to determine a feasible schedule while another cannot, we can conclude that the former is superior. Hence, the performance of the algorithms and parameter settings are compared using the $SuccessRatio\ (SR)$:

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

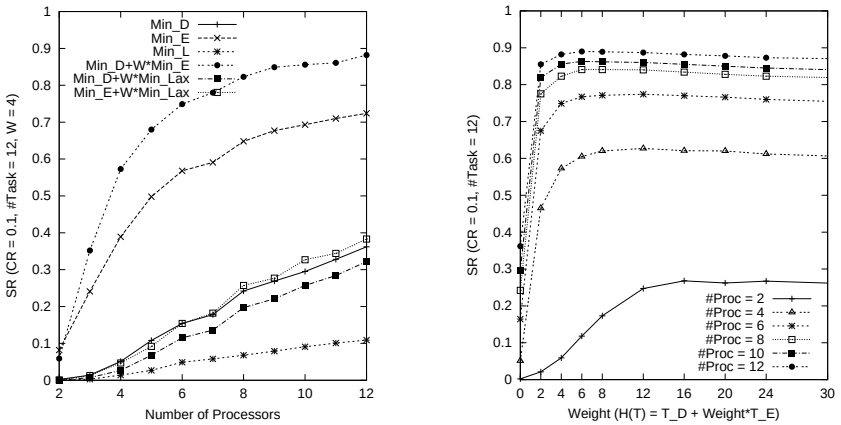
N^{succ} is the total number of schedulable task sets found by the algorithm, and N is the total number of task sets tested. Here N is 100 for each simulation run. Therefore for each result point in the graphs, $SR = (\sum_{i=1}^{10} SR_i)/10$, where $SR_i = N^{succ}/100$.

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.

Since there are several issues to be evaluated, we first study the heuristic functions to choose the best one and then explore other factors that affect the algorithm using this heuristic function.

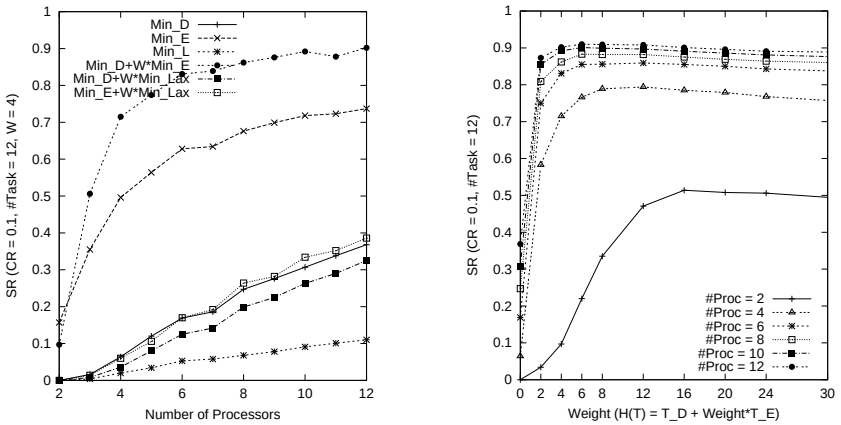
A. Choosing Scheduling Heuristics

Scheduling heuristics are based on the characteristics of tasks. We explore how different heuristic functions can affect finding a schedule. First, we investigate the sensitivity of each heuristic to the number of processors. Results are shown in Figure 8(a) and 9(a) for the greedy and aggressive algorithms respectively. As illustrated, for both algorithms, Min_E works much better than other simple heuristics. This is because after constructing the comprehensive graph, the earliest start time of each instance of the task encodes the basic precedence information. And from the beginning, Min_E chooses the best task that can be scheduled; it also maintains the precedence constraints. Min_L is not a good heuristic since the SR remains low even when the number of processors increases. However, Min_L might work better than other simple heuristics [19] if there are no precedence



(a) Effect of Heuristic Functions (b) Effect of Weight on the "Best" heuristic

Fig. 8. Effect of Heuristic and Weight for Greedy Algorithm



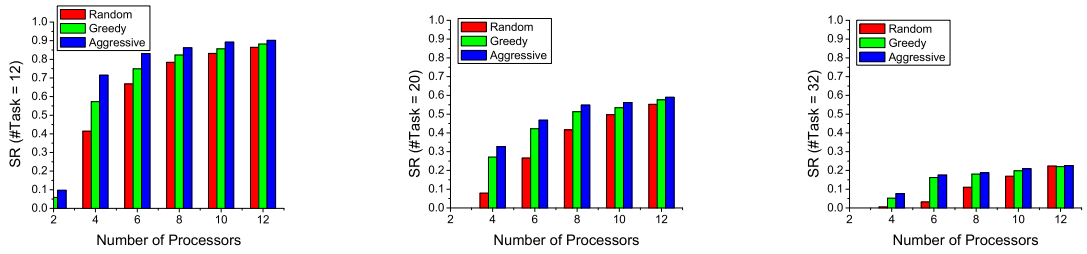
(a) Effect of Heuristic Functions (b) Effect of Weight on the "Best" heuristic

Fig. 9. Effect of Heuristic and Weight for Aggressive Algorithm

constraints. Therefore, the task set with precedence constraints behaves differently from an independent task environment, and Min_L is inadequate in such dependent environments.

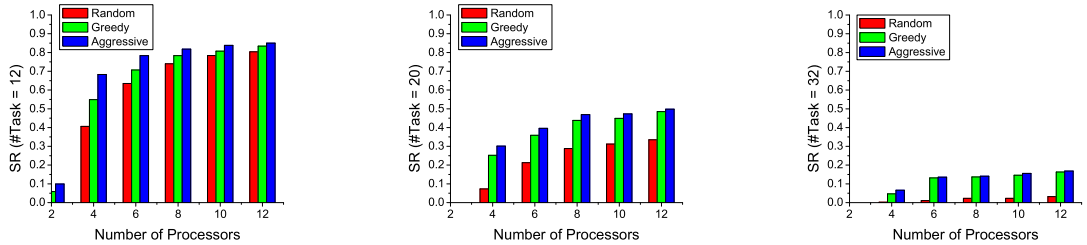
Now let us focus on the integrated heuristics. We find that $Min\_D + W * Min\_E$ has substantially better performance than other heuristics including the best simple heuristic Min_E. The reason should be clear: beside precedence constraints, deadline, an important temporal characteristic, is also taken into account in this heuristic function. Even though the other two integrated heuristics also take the deadline and earliest start time into consideration, they are more complex to compute, and their performance is not as good as $Min\_D+W*Min\_E$. Therefore, we choose $Min\_D + W * Min\_E$ as the scheduling heuristic for the following evaluations.

Since integrated heuristics are weighted combination of simple heuristics, we investigate the sensitivity of the best integrated heuristic, $Min\_D + W * Min\_E$, to changes in weight values. In Figure 8(b) and 9(b), we show one instance of the results with 12 tasks and *CR* of 0.1 for the greedy and aggressive algorithms separately. Hereafter, we denote "*The Number of Processors*" as "*#Proc*", and "*The Number of Tasks Within a Set*" as "*#Task*". When the weight is 0, the heuristic becomes the simple heuristic Min_D and does not perform well.



(a) $\#Task = 12$ (b) $\#Task = 20$ (c) $\#Task = 32$

Fig. 10. Effect of Heuristic Allocation Algorithms with $CR = 0.1$



(a) $\#Task = 12$ (b) $\#Task = 20$ (c) $\#Task = 32$

Fig. 11. Effect of Heuristic Allocation Algorithms with $CR = 0.4$

When the weight increases from 0 to 4, or from 0 to 12 when $\#Proc = 2$, we see a signifi cant 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 $\#Proc = 2$).

Another observation is that we see a similar profi le in performance for all number of processors, where there is an initial increase to a peak, and then performance decreases slightly as the weight increases. The reason is that when the weight becomes larger, the performance is affected more by Min_E. Therefore, the performance tends to be more like the performance with the simple heuristic Min_E.

B. Performance of the Allocation Algorithm

In this section, we test the performance of the greedy and aggressive technique that base their decisions on communication costs and processor workloads. Figures 10 and 11 show the results compared to those of the random allocation algorithm with $CR = 0.1$ and 0.4, respectively. The heuristic function used is $H(T) = Min\_D + W * Min\_E, W = 4$. Results are shown for three different task set sizes that have been tested with different numbers of processors. For a given task set with n tasks and m processors, we fi rst execute the period assignment algorithm to fi nd 100 such task sets that satisfy $\sum_{i=1}^{n}(C_i/P_i) \leq m$, and then apply greedy, aggressive, and random approaches to allocate tasks to see how many feasible schedules can be found. The six graphs illustrate that for all task set sizes with certain number of processors, the greedy heuristic performs better than the random allocation, while the aggressive heuristic performs better than the greedy method. Therefore, instead of assigning tasks randomly, exploiting inherent task characteristics and task relationships does help to fi nd feasible information driven allocations.

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 III
IMPROVEMENT OF GREEDY OVER RANDOM APPROACH (PERCENTAGE)

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

(a)$\#Task = 20$

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

(b) $\#Task = 32$

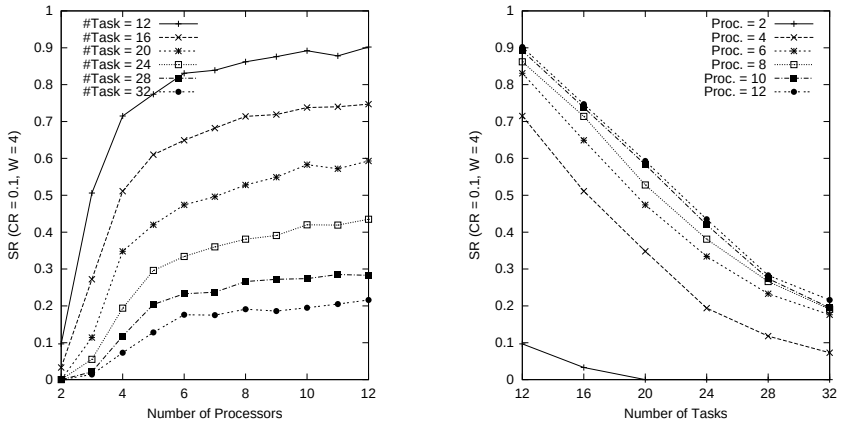
TABLE IV
IMPROVEMENT OF AGGRESSIVE TO RANDOM APPROACH (PERCENTAGE)

Comparing these figures, we find that for a certain number of processors and task set size, the improvement in performance of the greedy or aggressive strategy 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. For instance, with 8 processors and a task set size of $\#Task = 20$, the difference between the greedy (aggressive) and random algorithm is 9.7% (13.2%) when $CR = 0.1$, and the difference is 15% (18.1%) when $CR = 0.4$. In Table III, we show the difference in improvement of the greedy strategy compared to the random algorithm with a task set size of $\#Task = 20$ and $\#Task = 32$ respectively. Table IV shows the difference between the aggressive heuristic and the random method. 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 algorithm. 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, so it is not surprising that our heuristic based algorithms work better in a resource tight environment.

C. Effect of the Number of Processors and Tasks

In this section, we investigate the effects of processors and tasks. We investigate the performance of the algorithm under different #Proc values and #Task to show how they affect the schedulability. Here we only show evaluation results of the aggressive heuristic since it works better than the greedy heuristic. Figure 12(a) shows the improvement in performance with increasing number of processors and Figure 12(b) shows the drop in performance with increasing task set size. Both of them use the heuristic function of $Min\_D + W * Min\_E$.

For the effect of processors, it should be easy to understand that in normal cases, the larger the number of processors, the greater the improvement in performance. We also notice that for a given task set size, there might exist a key value for the number of processors. From $\#Proc = 2$ to that point, the performance improves



(a) Effect of Number of Processors (b) Effect of Task Set Size

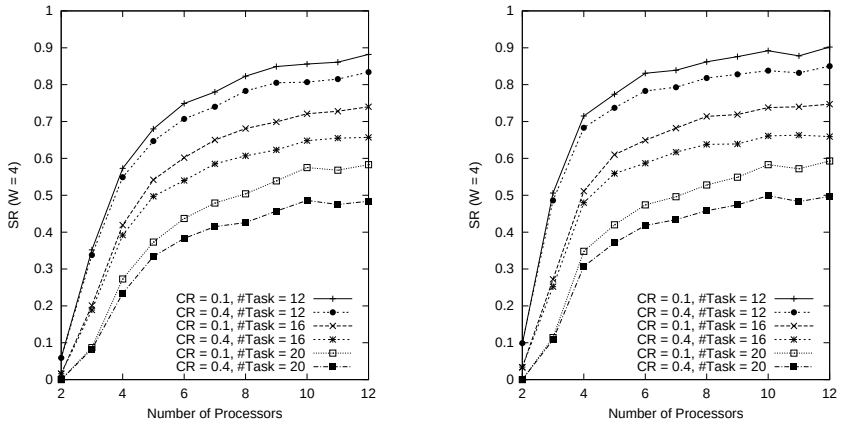
Fig. 12. Effect of Processors and Tasks

substantially, and after that point, the increase of the performance tends to be stable. For example, the key points are 6 and 5 for task set sizes of 12 and 16 respectively. This causes us to make an interesting observation: with some other resource, such as a communication channel, becoming the limit, the number of processors appropriate for a given task set will change.

Now let us take a look at the effect of the task set size. Given a processor set, when the task number increases, the success ratio generally decreases. The reason should be clear in that the more tasks, the higher the workload in the system. But we also find that given a fixed number of processors, the performance drops sharply until a certain task set size, after which it decreases more slowly as the task set size increases. This observation further shows that the schedulability is more sensitive to the integrated effect of processors and tasks than the single effect of the number of processors or the task set size. The results therefore suggest that when we design a system, we need to exploit the different types of resources and take into account all the factors that may explicitly or implicitly affect the performance.

D. Effect of Communication

Finally, we investigate the effect of another important resource — communication. The algorithm uses the greedy or aggressive allocation approaches to minimize the total communication cost by allocating two tasks to the same site. However, in case where two communicating tasks have to be placed on different sites, the communication cost becomes a very important factor in overall performance. Figure 13 shows the results of communication ratios of 0.1 and 0.4 with respect to the heuristic function of $Min\_D + W * Min\_E$ for three different task set sizes. As we can see from the graphs, when the number of processors is very limited, e.g., 2 or 3, the performance of the two cases is almost the same. This is because in both cases, it is hard to find a feasible schedule. But by increasing the number of processors, the performance with $CR = 0.1$ is better than with $CR = 0.4$. Since each communication introduces two additional precedence constraints that affect the earliest start time of consumers, and the communication costs affect the whole workload of the system, lowering communication leads to improved schedulability.



(a) Effect in Greedy Heuristic (b) Effect in Aggressive Heuristic

Fig. 13. Effect of Communication

VI. Application of our Strategy to Mobile Robotics

In this section, we return to the robotic problem discussed in Section I. The Worst Case Execution Times (WCET) of tasks are taken from an experimental implementation on a StrongARM 206MHz CPU; communication costs are based on the bytes transmitted using 802.11b wireless protocol with 11 Mbit/s transmission rate. The figures are given in Tables V and VI respectively. The periods are 220 ms for all sensor tasks and motor drivers. Therefore, the periods of controller tasks are also assigned to be 220 ms by the harmonic constraint.

A. Allocation and Scheduling Decisions

First, let us use the aggressive heuristic to analyze the location of tasks executing the push policy. The starting utilizations of the two processors are the same since initially they have the same hardware-related sensor tasks. So $u_1 = u_2 = 120/220 + 20/220 + 20/220 = 0.7273$. Hence the threshold $t = 0.7273$. Then the algorithm works as follows:

1. Consider H_1. Since the sum of CCRs from $site1$ is larger than that from $site2$, and all sites have the same utilization, H_1 is assigned to S^1, and $t = u_1 = 0.8864$.
2. Consider H_2. Since all communication costs are from tasks located on $site2$, and the utilization of $site2$ is less than the threshold if H_2 is assigned to S^2, H_2 is allocated to S^2, and the threshold is still $t = 0.8864$;
3. Finally, consider task L_2. Since $CCR_{H_2,L_2} > CCR_{H_1,L_2}$, and the threshold will be less than the current value if L_2 is assigned to S^2, the algorithm will locate L_2 on S^2.

After assigning tasks, only two communication costs need be consisdered: $IR_2 \rightarrow H_1$ (0.02327) and $H_1 \rightarrow L_2$ (2.979). The earliest start time of L_2 is $max(\{POS_2, IR_2\} \rightarrow H_2 \rightarrow L_2, \{POS_1, IR_1, IR_2\} \rightarrow H_1 \rightarrow L_2) = max(165, 178.00227) = 178.00227$ (ms). Hence the task set is schedulable by $Min\_D + Min\_E$.

B. Prediction of Scalability

Now let us take a look at the team with three robots. The task and communication models are shown in Figure 4(a). All tasks capture the same characteristics as two robots except the communication to or from $site3$.

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 V
WCETS(MS) OF TASKS IN FIGURE 2

Comm.	$IR_{1(2)} \rightarrow H_{1(2)}$	$Pos_{1(2)} \rightarrow H_{1(2)}$	$H_1 \rightarrow L_2$	$H_1 \rightarrow M_1$	$H_2 \rightarrow L_2$	$Pos_1 \rightarrow L_2$	$L_2 \rightarrow M_{2(1)}$
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 VI
COMMUNICATION COSTS OF FIGURE 2

Following the same reasoning as two robots, the algorithm allocates H_3 and L_3 to S^3.

When we design a collaborative robot team with the same type of robotic members, each robot will possess the same task charateristics. Besides, if the communication pattern is similar, as in the *push* chain, a larger team can be dealt with simply by generalizing the results from the smaller team, *e.g.*, a two-robot pair, Leader/Follower. This is because our allocation strategy captures and takes into account task characteristics, communication pattern and workloads together. Therefore, our approach leads to predictable scalability analysis.

The only problem left is to study how the use of the shared communication channel scales for a larger team. This can be analyzed based on the critical-path-time C_{max}, the maximum sum of execution times of all tasks, including communicaiton tasks, along any path from an (a set of) input task(s) to an output task in the task graph. For a team with n robots as in our push model, if n is large enough, C_{max} is calculated from the path: $\{POS_1, IR_1, IR2\} \rightarrow H_1 \rightarrow L_2 \rightarrow L_3 ... \rightarrow L_n \rightarrow M_n$, where M_n is the motor task on the n^{th} site. Therefore, we can calculate the upper bound for the number of robots in such a team effi ciently. For instance, considering the push model for two robots, the C_{max} is $120 + 20 + 0.02327 + 35 + 2.979 + 5 + 20 = 203.00227\ (ms)$, the laxity time within a period is $220 - 203.00227 = 16.99773\ (ms)$. When the team member increases, the computation time from $L_i \rightarrow L_{i+1}$ is introduced. Hence, the upper bound is $n = \lfloor 16.99773 \div (5 + 2.979) \rfloor + 2 = 4$, where the laxity for Site 4 is $1.03973\ ms$. Generally speaking, if we know that a team of size n is schedulable, and since we know the effect on the computation time along the critical path of the $n + 1^{st}$ robot, we can compare this effect with the laxity of the n robot team to see if the team with $n + 1$ members is schedulable. This is the case if the laxity is larger than the sum of additional communication cost and additional task execution time (s).

If during the design phase, we fi nd that the number of robots in a team is larger than n, the upper bound, we can split the team into small groups geographically at run time, and each group is a high level robot in which only one robot needs to communicate with a robot in another group to share the information and/or decisions. Even though the communication resource is limited, we can know in advance how many resources will be free for a small group given the pre-analysis done by our algorithm. For the example in our LOS communication chain, the building blocks of a small robot group are *push-push, push-pull, pull-push* and *pull-pull.* How many groups

are needed depends on the run time environment and team size. But once we have the allocation assignment and the scheduling bound for each building block, optimal combinations of groups with different sizes can be generated as a result of our algorithm. At run time a dedicated hierarchical communication model can be built just by looking up the grouping of the robots.

VII. Conclusion

Allocating and scheduling of real-time tasks in a distributed environment is a difficult problem. In addition to task-level constraints, *e.g.*, periods and deadlines, such systems also have system-level constraints, *e.g.*, precedence, communication and locality. The algorithm discussed in this paper provides a framework for allocating and scheduling periodic tasks in distributed embedded systems, in which the inherent scale and symmetric properties, such as the LOS chain of a multi-robot system, are well captured by our allocation methods.

In the scheduling part of the algorithm, various temporal characteristics of tasks are taken into account at each search step. We evaluated the algorithm by varying several factors that could explicitly or implicitly affect the performance of the system. Also, the algorithm was exercised 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 this approach can enable the system developers to design a predictable distributed embedded systems even if it has a variety of temporal and resource constraints.

Now we discuss some of the possible extensions to the algorithm. First, even when the system design does not constrain the locality of input/output tasks in advance, the heuristic allocation algorithm can still be used. In this case, the initial load threshold is 0 since no processor has loaded any task. After selecting the first pair of communicating tasks that have the largest communication cost ratio and assigning them to a randomly selected processor, the algorithm can continue to allocate other tasks as discussed above.

Second, the algorithm can be extended 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 as shown in the algorithm. Second, when select the processor, if the task can be assigned to the processor that the producer is on, 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.

Finally, the algorithm can be tailored to apply to systems that go through different modes [17]. For instance, a third robot could join a two-robot leader/follower team and become a follower or the roles of leader/follower changes. Such a situation can be modeled as a mode change. Constraints can be imposed on the original schedules constructed in anticipation of the arrival of robots into the team. In order to maintain continuity of allocation of tasks and schedulability, we can do further off-line schedulability analysis for different workloads under possible equivalent loads in different modes. Hence at run time, the system refers to a tabular representation of feasible allocations and schedules.

ACKNOWLEDGEMENTS

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

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

[14] 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.

[15] 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.

[16] 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.

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

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

[19] K. Ramamritham, J. A. Stankovic, and P. Shiah. Effi cient scheduling algorithm for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):184–194, April 1990.

[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 Canadians 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. Accepted for publication.*

[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.

APPENDICES

Input, Output and variables of Allocation Algorithms

Input: a task graph $G = (E, V)$ with period and worst case execution time for each task;
communication costs for all pairs of communicating tasks;
the locality constraints for input and output tasks;
the number of processors 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 $\frac{communication\_cost(T_i \rightarrow T_j)}{C_i + C_j}$

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

TABLE VII
NOTATIONS FOR ALLOCATION ALGORITHMS

Function of Assignment and Threshold Update

int **thresholdUpdate(float** $u_k^{'}$**, int** k**, Task** T_y**)**

/* t is the threshold, T_y and processor k is selected, $u_k^{'}$ is the new workload if assinging T_y to k */

1. **Case 1:** $u_k^{'} \leq t$, **do** /* $u_k^{'}$ is less than the threshold t */
2. Assign task T_y to processor k;
3. Update U with the new utilization $u_k = u_k^{'}$;
4. **Case 2:** $u_k^{'} > t$, **do** /* $u_k^{'}$ 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:** $u_k^{'} > 1$, **do** /* processor k cannot take 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:** $u_k^{'} \leq 1$, **do** /* $u_l^{'} \leq u_k^{'} \leq 1$ */
13. Allocate task T_y to processor l;
14. Update U with the new $u_l = u_l^{'}$;
15. $t = u_k^{'}$;
16. **Return** t; /* new threshold */

TABLE VIII

FUNCTION OF ASSIGNMENT AND THRESHOLD UPDATES

Greedy Allocation Algorithm

1. Initialize $U = \{u_i | i = 1, 2, ..., m\}$, 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 $R = \{CCR_{x,y} | T_x \in F, T_y \in N\}, T_x \rightarrow T_y$;
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 processor k */
9. **If** $((t = thresholdUpdate(u_k^{'}, k, T_y)) < 0)$, **do**;
10. exit without solution; /* cannot find a processor */
11. Update set F, N such that $F = F \cup \{T_y\}, N = N \setminus \{T_y\}$;
12. Update set R, such that $R = R \setminus \{CCR_{x,y}\} \cup \{CCR_{y,z}\}, \forall T_x \in F, T_z \in N(T_x \rightarrow T_y) \wedge (T_y \rightarrow T_z)$.

TABLE IX

GREEDY ALLOCATION ALGORITHM