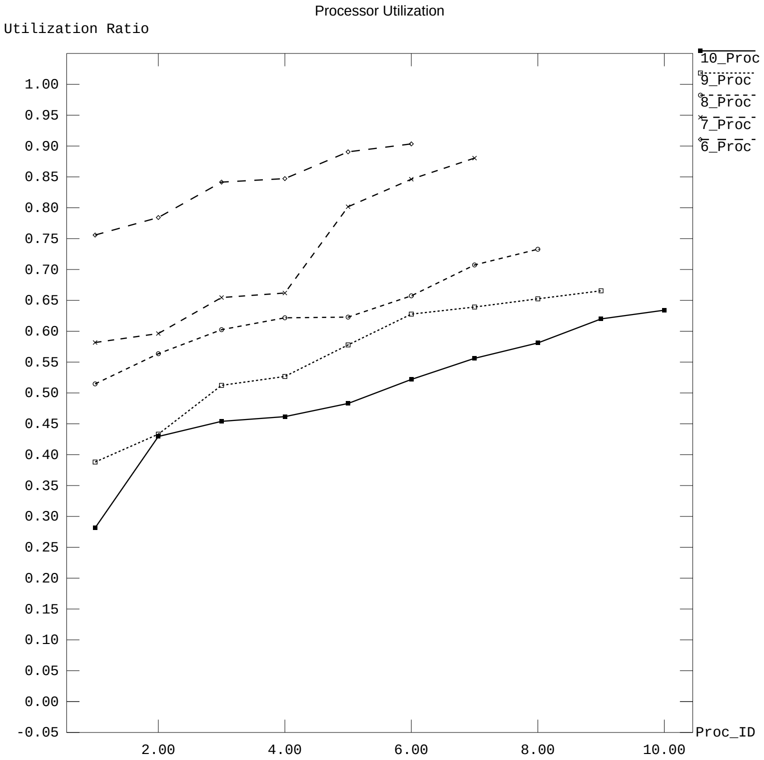


Figure 8: Processor Utilization Ratios for different cases

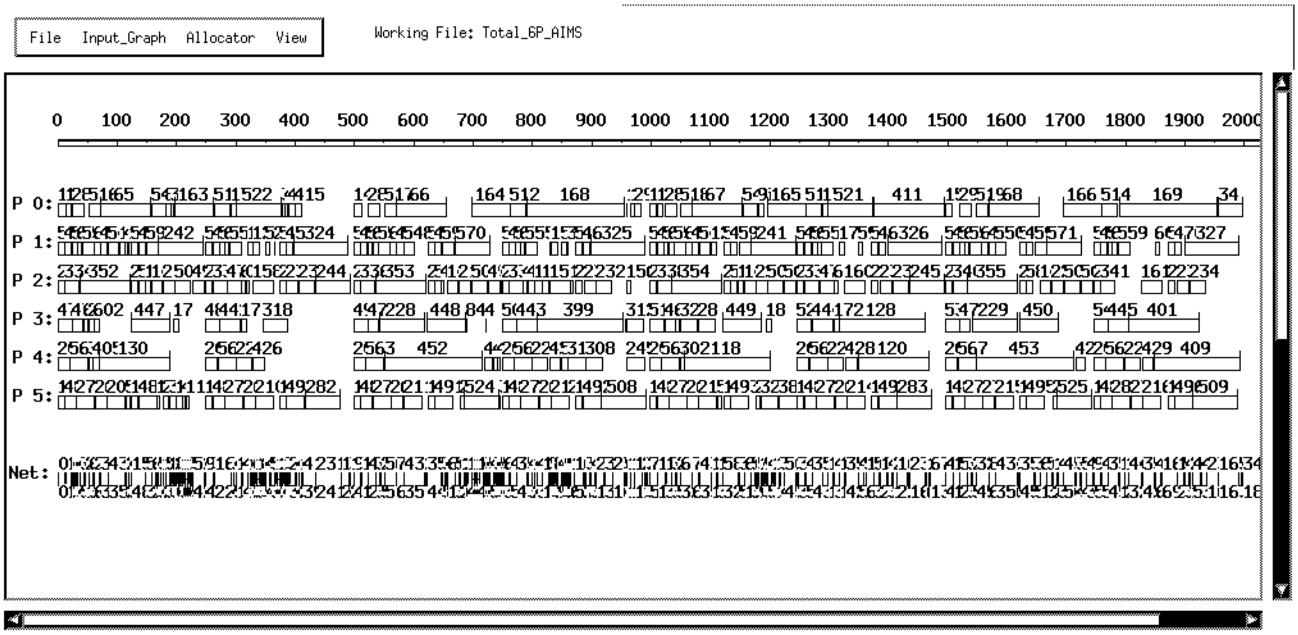


Figure 9: The Allocation Results and Schedules for AIMS with 6 processors

```
Choose an initial temperature $T$
Choose randomly a starting point $P = (\phi, \sigma_m, \sigma_c)$
$E_p$ := Energy of solution point $P$
if $E_p = 0$ then
    output $E_p$ and exit /* $E_p = 0$ means a feasible solution */
end if
repeat
    repeat
        Choose $N$, a neighbor of $P$
        $E_n$ := Energy of solution point $N$
        if $E_n = 0$ then
            output $E_n$ and exit /* $E_n = 0$ means a feasible solution */
        end if
        if $E_n < E_p$ then
            $P := N$
            $E_p := E_n$
        else
            $x := \frac{E_p - E_n}{T}$
            if $e^x \geq$ random(0,1) then
                $P := N$
                $E_p := E_n$
            end if
        end if
    until thermal equilibrium at $T$
    $T := \alpha \times T$ (where $\alpha < 1$)
until stopping criterion
```

Figure 7: The structure of simulated annealing algorithm.

```
Given a solution point $P = (\phi, \sigma_m, \sigma_c)$
While there is some unscheduled task instance do
    Find the next unscheduled instance. /* By the SLsF algorithm */
    Let the instance be $\tau_i^j$.
    Sort all the incoming communications of $\tau_i^j$ based on the latency values into a descending order.
    Schedule each incoming communication starting from the biggest-latency one to the tightest-latency one.
    Schedule the instance $\tau_i^j$.
End While.
Mark each instance as un-examined.
While there is some un-examined task instance do
    Find the next un-examined task instance. /* By the finish times */
    Sort all the outgoing communications of the task instance based on the latency values into an increasing order.
    Schedule each outgoing communication starting from the tightest-latency one to the biggest-latency one.
    Mark the task instance examined.
End While.
Collect the start time and finish time informations for each task instance and communication.
Compute the energy value using Equation 5.
```

Figure 6: The pseudo code for computing the energy value

[CDHC94] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini. Arinc 659 scheduling: Problem definition. In *Proceedings of IEEE Real-Time Systems Symposium*, San Juan, PR, Dec. 1994.

[GMK+91] Ó. Gudmundsson, D. Mossé, K.T. Ko, A.K. Agrawala, and S.K. Tripathi. Maruti: A platform for hard real-time applications. In K. Gordon, A.K. Agrawala, and P. Hwang (eds.), editors, *Mission Critical Operating Systems*. IOS Press, 1991.

[HS92] Chao-Ju Hou and Kang G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. In *Proceedings of the 1992 IEEE 13th Real-Time Systems Symposium*, pages 146–155, Phoenix, AZ, 1992.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[LH91] Feng-Tse Lin and Ching-Chi Hsu. Task assignment problems in distributed computing systems by simulated annealing. *Journal of the Chinese Institute of Engineers*, 14(5):537–550, Sept. 1991.

[MSA92] Daniel Mossé, M.C. Saksena, and Ashok K. Agrawala. Maruti: An approach to real-time system design. Technical Report CS-TR-2845, UMIACS-TR-92-21, Department of Computer Science, University of Maryland, College Park, 1992.

[Ram90] Krithi Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 108–115, Paris, France, 1990.

[SdSA94] M. Saksena, J. da Silva, and A. K. Agrawala. Design and implementation of *maruti-ii*. Technical Report CS-TR-2845, Department of Computer Science, University of Maryland, College Park, 1994.

[TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: an NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992.

5.1 Discussions

For feasible solutions of the AIMS with various numbers of processors, we calculate the processor utilization ratio (PUR) of each processor. The processor utilization ratio for a processor p is defined as

$$\frac{\sum_{\phi(t_i)=p}(e_i \times q_i)}{LCM}.$$

The results are shown in Figure 8. The ratios are sorted into a non-decreasing order given a fixed number of processors. The algorithm generates the feasible solutions for the AIMS with 6, 7, 8, 9 and 10 processors respectively. For example, for the 6-processor case, the PURs for the heaviest-loaded and lightest-loaded processors are 0.91 and 0.76 respectively. For the 10-processor cases, the PURs are 0.63 and 0.28 respectively. We find that the ratio difference between the heaviest-loaded processor and the lightest-loaded processor in the 6-processor case is smaller than those in other cases. It means the chance for a more load-balanced allocation to find a feasible solution is bigger when the number of processors is smaller.

The detailed schedules for the 6-processor case are shown in Figure 9. The results are shown on an interactive graphical interface which is developed for the design of *MARUTI*. The time scale shown in Figure 9 is 100μs. So the LCM is shown as 2000 in the figure. (i.e. $2000 \times 100\mu s = 200ms$.) This solution consists of seven off-line non-preemptive schedules: one for each processor and one for the SafeBus (TM). Each of these schedules will be one LCM long where an infinite schedule can be produced by repeating these schedules indefinitely. Note that the pseudo instances are introduced to make sure the wrapping around at the end of the LCM-long schedules should satisfy the latency and next-execution-interval requirements across the point of wrap-around. The pseudo instances are not shown in Figure 9.

The inclusion of resource and memory constraints into the problem can be done by modifying neighbor-finding strategy. Once a neighbor of the current point is generated, it is checked to ascertain that the constraints on memory etc. are met. If not, the neighbor is discarded and another neighbor is evaluated.

References

[CA93] Sheng-Tzong Cheng and Ashok K. Agrawala. Scheduling of periodic tasks with relative timing constraints. Technical Report CS-TR-3392, UMIACS-TR-94-135, Department of Computer Science, University of Maryland, College Park, December. 1993. Submitted to *the 10th Annual IEEE Conference on Computer Assurance, COMPASS '95.*

	10_Proc	9_Proc	8_Proc	7_Proc	6_Proc
Exec_Time (Sec)	2369	5572	19774	36218	78647
$= Hr : Min : Sec$	0:39:29	1:32:52	5:29:34	10:03:38	21:50:47

Table 1: The execution times of the AIMS with different number of processors

system connected by a SafeBus (TM) ultra-reliable bus. The problem is to find the minimum number of processors needed to assign the tasks to these processors. The objective is to develop an off-line non-preemptable schedule for each processor and one schedule for the SafeBus (TM) ultra-reliable bus.

The AIMS consists of 155 tasks and 951 communications between these tasks. The frequencies of the tasks vary from 5HZ to 40HZ. The execution times of the tasks vary from $0ms$ to $16.650ms$. The NEI and XEI of a task t_i are $p_i - 500\mu$s and $p_i + 500\mu$s respectively. Since $\delta = 1000\mu$s $= 1ms < \frac{25ms}{e}$, the smallest-period-first scheduling algorithm can be used in this case. Tasks communicate with others asynchronously and in mutuality. The transmission times for communications are in the range from 0μs to 447.733μs. The latency constraints of the communications vary from $68.993ms$ to $200ms$. The LCM of these 155 tasks is $200ms$. When the whole system is extended, the total number of task instances within one scheduling frame is 624 and the number of communications is 1580.

For such a real and tremendous problem size, pre-analysis is necessary. We calculate the resource utilization index to estimate the minimum number of processors needed to run AIMS. The index is defined as

$$\frac{\sum_{i=1}^{155}(e_i \times q_i)}{LCM}$$

where e_i is the execution of task t_i and $q_i = \frac{LCM}{p_i}$. The obtained index for AIMS is 5.14. It means there exist no feasible solutions for the AIMS if the number of processors in the multiprocessor system is less than 6.

The number of processors which the AIMS is allowed to run on is a parameter to the scheduling problem. We start the AIMS scheduling problem with 10 processors. After a feasible solution is found, we decrease the number of processors by one and solve the whole problem again. We run the algorithm on a DECstation 5000. The execution time for the AIMS scheduling problem with different numbers of processors is summarized in Table 1. The algorithm is able to find a feasible solution of the AIMS with six processors which is the minimum number of processors according to the resource utilization index. The time to find such a feasible solution is less than one day (approximately 22 hours).

- Balance Mode: We randomly move a task from the heavily-loaded processor to the lightest-loaded processor. This move tries to balance the workload of processors. By balancing the workload, the chance to find a neighbor with a lower energy value is bigger.
- Swap Mode: We randomly choose two tasks τ_i and τ_j on processors p and q respectively. Then we change ϕ by setting $\phi(\tau_i) = q$ and $\phi(\tau_j) = p$.
- Merge Mode: We pick two tasks and move them to one processor. By merging two tasks to a processor, we increase the workload of the processor. There is an opportunity of increasing the energy level of the new point by increasing the workload of the processor. The purpose of the move is to perturb the system and allow the next move to escape from the local optimum.
- Direct Mode: When the system is in a low-energy state, only few tasks violate the jitter or latency constraints. Under such a circumstance, it will be more beneficial to change the assignment of these tasks instead of randomly moving other tasks. From the conducted experiments, we find that this mode can accelerate the searching of a feasible solution especially when the system is about to reach the equilibrium.

The selection of the appropriate mode to find a neighbor is based on the current system state. Given a randomly generated initial state (i.e. solution point), the workload discrepancy between the processors may be huge. Hence, in the early stage of the simulated annealing, the balance mode is useful to balance the workload. After the processor workload is balanced out, the swap mode and the merge mode are frequently used to find a lower energy state until the system reaches near-termination state. In the final stage of the annealing, the direct mode tries to find a feasible solution. The whole process terminates when a feasible solution is found in which the energy value is zero.

5 Experimental Results

We implemented the algorithm as the framework of the allocator on *MARUTI* [GMK$^+$91, MSA92, SdSA94], a real-time operating system developed at the University of Maryland, and conducted extensive experiments under various task characteristics. The tests involve the allocation of real-time tasks on a homogeneous distributed system connected by a communication channel.

To test the practicality of the approach and show the significance of the algorithm, we consider a simplified and sanitized version of a real problem. This was derived from actual development work, and is therefore representative of the scheduling requirements of an actual avionics system. The Boeing 777 Aircraft Information Management System (AIMS) is to be running on a multiprocessor *time* of the communication. Once the time slot is inserted, we check the *effective start time* of τ_i^j to make sure that it is not less than the *finish time* of the time slot. If it is, the *effective start time* of τ_i^j is updated to be the *finish time* of the time slot.

If a task instance has more than one incoming communication, the scheduling order among these communications is based on their latency constraints. The bigger the latency value is, the earlier the communication is scheduled. The incoming communication with the tightest latency constraint is scheduled last. It is because the *effective start time* of the receiving task instance is constantly updated by the scheduling of the incoming communications. It is possible that the scheduling of the later incoming communications increases the *effective start time* of the receiving task instance and make the early scheduled communication violate its latency constraint if the constraint is tight.

4.1.3 Scheduling the Outgoing Communications: σ_c

The scheduling of the outgoing communications for the whole task set is performed after all the task instances have been scheduled. The scheduling order among these communications is based on the finish times of the sending task instances. The task instance with the smallest finish time is considered first. When a task instance is taken into account, all its outgoing communications are scheduled one by one according to their latency constraints. The communication with the tightest latency constraint is scheduled first.

Given an outgoing communication $\tau_i^j \mapsto \tau_k^?$, and the finish time of τ_i^j, f_i^j, the *effective start time* of the communication is set to be f_i^j. Based on the *effective start time*, a time slot in inserted for this communication. Then the nearest instance of receiving task can be found based on the *finish time* of the time slot.

For the example shown in Figure 5, The incoming communication marked with "(1)" is scheduled before the scheduling of τ_y^2. The sixth instance of τ_x is chosen as the nearest instance. As for the outgoing communication marked with "(3)", it is scheduled after the scheduling of τ_x^5, τ_x^6, τ_x^7, and τ_x^8. In this example, τ_x^8 is the nearest instance of the outgoing communication.

4.2 Neighbor Finding Strategy: ϕ

The neighbor finding strategy is used to find the next solution point once the current solution point is evaluated as infeasible (i.e. energy value is nonnegative). The neighbor space of a solution point is the set of points which can be reached by changing the assignment of one or two tasks. There are several modes of neighbor finding strategy.

4.1.1 Priority Assignment of Task Instances: σ_m

In the work [CA93], we presented the SLsF algorithm and the performance evaluation. The results showed that SLsF outperforms SPF and SJF. In this paper we use the SLsF as the priority assignment algorithm for the task instances in I.

Formally, if $lst(\tau_i^j) < lst(\tau_k^\ell)$, then $\sigma_m(\tau_i^j) < \sigma_m(\tau_k^\ell)$. And the insertion of a time slot for τ_i^j precedes that for τ_k^ℓ if $\sigma_m(\tau_i^j) < \sigma_m(\tau_k^\ell)$. The time-based scheduling algorithm for a task instance is used to find a time slot for a task instance once the *effective start time* is given. We define the *effective start time* of a task instance as the earliest start time when the incoming communications are taken into account. Let t be the maximum completion time among all the incoming communications of a task instance, then the *effective start time* of the task instance is set to the bigger value among t and est (as stated in Equation 6).

4.1.2 Scheduling the Incoming Communications: σ_c

There are two kinds of incoming communications. The first kind is called the synchronous communication in which the frequencies of the sender and receiver are identical. The other kind is called the asynchronous communication in which the sending task instance is associated with a question mark. For such an asynchronous communication, we have to decide which instance of the sending task should communicate with the receiving task instance. The approach we take is to find the nearest instance of the sending task. The reason is that, by finding the nearest instance, the time difference between start time of the receiving instance and the completion time of the sending instance is the smallest. The chance of violating the latency constraint of a communication will be the smallest then.

The nearest instance of a sending task can be found using the following method. Given an incoming communication $\tau_k^? \mapsto \tau_i^j$, and the *effective start time* of τ_i^j, eft we search through the linked list of processor $\phi(\tau_k)$ up to time eft. If there is some instance of τ_k, say τ_k^ℓ, whose completion time is the latest among all scheduled instances of τ_k, then the nearest instance is found. Otherwise, we continue to search through the linked list until an instance of τ_k is found. We set the *effective start time* of the communication to be the completion time of the found instance. We also erase the question mark such that $\tau_k^? \mapsto \tau_i^j$ is changed to $\tau_k^\ell \mapsto \tau_i^j$. For the synchronous communication, the *effective start time* of the communication is simply assigned as the *finish time* of the sending task instance.

The scheduling of the communication is done by inserting a time slot to the linked list for the communications network. The *start time* of the time slot can not be earlier than the *effective start* the algorithm is to randomly choose an assignment ϕ, a total ordering of instances within one scheduling frame, σ_m, and a total ordering of communications for the instances, σ_c. A solution point in the search space of SA is a 3-tuple (ϕ,σ_m,σ_c). The energy of a solution point is computed by equation (5). For each solution point P which is infeasible, (i.e. E_p is nonzero), a neighbor finding strategy is invoked to generate a neighbor of P. As stated before, if the energy of the neighbor is lower than the current value, we accept the neighbor as the current solution; otherwise, a probability function (i.e. $exp(\frac{E_p - E_n}{T})$) is evaluated to determine whether to accept the neighbor or not. The parameter of the probability function is the current temperature. As the temperature is decreasing, the chance of accepting an uphill jump (i.e. a solution point with a higher energy level) is smaller. The inner and outer loops are for thermal equilibrium and termination respectively. The number of iterations for the inner loop is also a function of current temperature. The lower the temperature is, the bigger the number is. Methods about how to model the numbers of iterations and how to assign the number for each temperature have been proposed [LH91]. In this dissertation, we consider a simple incremental function. Namely, $N = N + \Delta$ where N is the number of iterations and Δ is a constant. The termination condition for the outer loop is $E_p = 0$. Whenever thermal equilibrium is reached at a temperature, the temperature is decreased. Linear or nonlinear approach of temperature decrease function can be simple or complex. Here we consider a simple multiplication function (i.e. $T = T \times \alpha$, where $\alpha < 1$).

4.1 Evaluation of Energy Value for a Solution Point (ϕ, σ_m, σ_c)

The computation of the energy value stated in Equation 5 , is done by constructing multi-processor schedules and a network schedule, and collecting the the start and completion times of each task instance and communication from these schedules.

The construction of the schedules is characterized by the priority assignment of the task instances in the set. The priority assignment algorithm determines the scheduling order among all the task instances. Each time when a task instance is chosen to be scheduled, the incoming communications of the instance are scheduled first and then the task instance itself. After all the task instances have been scheduled, the scheduling of the outgoing communications is performed. An algorithmic description about how to compute the energy value for a solution point is given in Figure 6. Note that a communication is an incoming communication to a task instance if the frequency of the receiving task instance is equal to or less than that of the sending task instance. For example, $\tau_k^? \mapsto \tau_i^j$ and $\tau_k^j \mapsto \tau_i^j$ are incoming communications to τ_i^j. On the other hand, if the sender frequency is less than the receiver frequency, then the communication is an outgoing communication. (e.g. $\tau_k^j \mapsto \tau_i^?$ is the outgoing communication of τ_k^j).

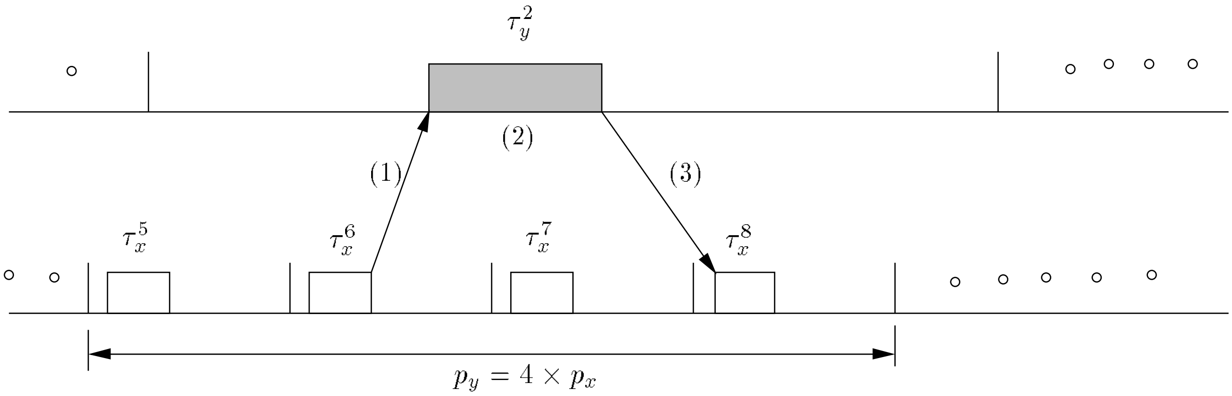


Figure 5: Asynchronous communications in mutuality

and k can be stated as

$$(j-1) \times n < i < k \leq j \times n. \tag{8}$$

A graphical illustration can be found in Figure 5. In the example, the values of i, j, k, and n are 6, 2, 8, 4 respectively. The communications $\tau_x^6 \mapsto \tau_y^2$ and $\tau_y^2 \mapsto \tau_x^8$ are scheduled before and after the scheduling of τ_y^2 respectively.

4 The Simulated Annealing Algorithm

Kirkpatrick *et al.* [KGV83] proposed a simulated annealing algorithm for combinatorial optimization problems. Simulated annealing is a global optimization technique. It is derived from the observation that an optimization problem can be identified with a fluid. There exists an analogy between finding an optimal solution of a combinatorial problem with many variables and the slow cooling of a molten metal until it reaches its low energy ground state. Hence, the terms about energy function, temperature, and thermal equilibrium are mostly used. During the search of an optimal solution, the algorithm always accepts the downward moves from the current solution point to the points of lower energy values, while there is still a small chance of accepting upward moves to the points of higher energy values. The probability of accepting an uphill move is a function of current temperature. The purpose of hill climbing is to escape from a local optimal configuration. If there are no upward or downward moves over a number of iterations, the thermal equilibrium is reached. The temperature then is reduced to a smaller value and the searching continues from the current solution point. The whole process terminates when either (1) the lowest energy point is found or (2) no upward or downward jumps have been taken for a number of successive thermal equilibrium.

The structure of simulated annealing (SA) algorithm is shown in Figure 7. The first step of

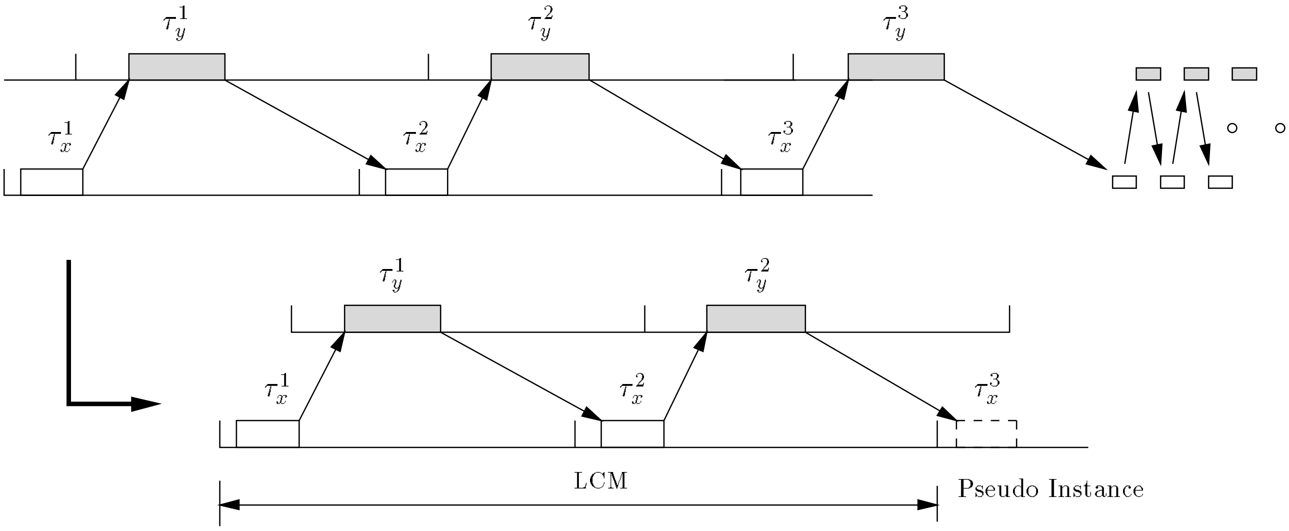


Figure 4: The introduction of a pseudo instance

and wrapped to the beginning of the schedule. As shown in Figure 3 The *start_time* of the new slot is r while the completion time is $r + e-$LCM.

3.3 Pseudo Instances

As stated in Section 2, we consider the communication pattern in which cyclic dependency exists among tasks. Given a set of tasks, Γ, a set of task instances, I, a set of communications, C, and any solution point, $(\phi, \sigma_m, \sigma_c)$, we introduce pseudo instances to solve this problem. For any task τ_x, if there exists a task τ_y, in which (1) $\sigma_m(\tau_x^i) < \sigma_m(\tau_y^i)$, $\forall\ i$, (2) $n_x = n_y$, and (3) $\tau_x \mapsto \tau_y \in C$ and $\tau_y \mapsto \tau_x \in C$, then a pseudo instance $\tau_x^{n_x+1}$ is added to I. A pseudo instance is always a receiving instance. No insertion of time slots for pseudo instances is needed. For a pseudo instance, only the effective start time is concerned. The effective start time of a pseudo instance $\tau_x^{n_x+1}$ in the constructed schedule based on $(\phi, \sigma_m, \sigma_c)$ is checked to see whether it is less than LCM + s_x^1 or not. If yes, then the execution of τ_x^1 for the next scheduling frame may start at $LCM + s_x^1$ which is exactly one LCM away from the execution of τ_x^1 for the current scheduling frame. A graphical illustration of the introduction of pseudo instance to solve the synchronous communications of cyclic dependency is given in Figure 4 in which $n_x = 2$.

As for the asynchronous communications of cyclic dependency, no pseudo instances are needed. For example, if both $\tau_x \mapsto \tau_y$ and $\tau_y \mapsto \tau_x$ exist and $n_x = n_y \times n$, then for each τ_y^j, where $j = 1$, 2, …, n_y, find a sending instance $\tau_x^i \in I$ and a receiving instance $\tau_x^k \in I$ such that (1) $f_x^i \leq s_y^j$, (2) $f_y^j \leq s_x^k$, and (3) $\tau_x^i \mapsto \tau_y^j$ and $\tau_y^j \mapsto \tau_x^k$ are the communications. The relationship between i, j,

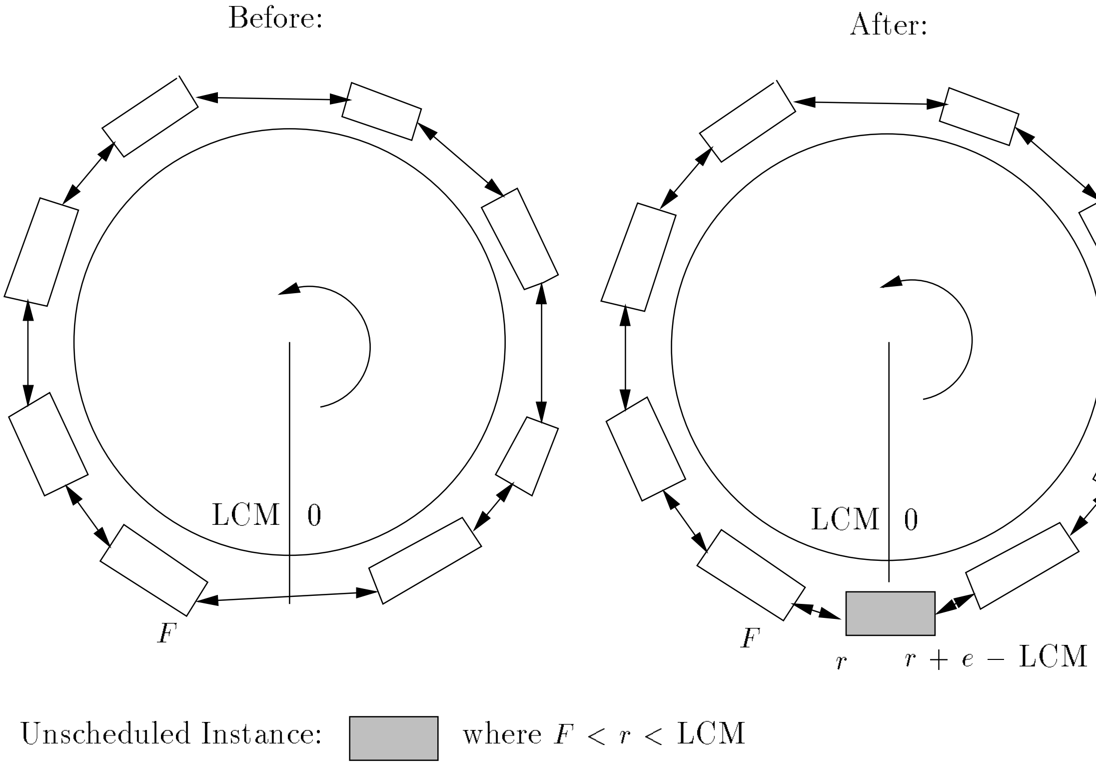


Figure 3: Insertion of a new time slot

3.2.1 Recurrence

Given any solution point $(\phi, \sigma_m, \sigma_c)$, we construct the schedule by inserting time slots to the linked lists. Let σ_m: *task_id* × *instance_id* → *integer*. The insertion of a time slot for τ_i^j precedes that for τ_k^ℓ if $\sigma_m(\tau_i^j) < \sigma_m(\tau_k^\ell)$.

Recall that Equations 6 and 7 specify the bounds of the scheduling window for a task instance. Due to the communications, $est(\tau_i^j)$ in Equation 6 may not be the earliest time for τ_i^j. We define the *effective start time* as the time when (1) the hybrid constraints are satisfied and (2) τ_i^j receives all necessary data or messages from all the senders.

Given the effective start time r and the assignment of τ_i (i.e. $p = \phi(\tau_i)$), a time slot of processor p is assigned to τ_i^j where $start\_time \geq r$ and $finish\_time - start\_time = e_i$. Note that we have to make sure the new time slot does not overlap existent time slots. Since (1) the executions of all instances within one scheduling frame recur in the next scheduling frame and (2) it is possible that the time slot for some instance is over LCM, we subtract one LCM from the $start\_time$ or $finish\_time$ if it is greater than LCM. It means the time slot for this task instance will be modulated

Equations 3 and 4. Formally, given s_i^1, s_i^2, ..., and ..., s_i^{j-1}, the problem is to derive the feasible scheduling window for τ_i^j such that a feasible schedule can be obtained if τ_i^j is scheduled within the window.

Proposition 1 [CA93]: Let the est and lst of τ_i^j be

$$\begin{aligned} est(\tau_i^j) &= max\{(s_i^{j-1} + p_i - \lambda_i),\ (s_i^1 + (j-1) \times p_i - (n_i - j + 1) \times \eta_i)\}, & (6) \\ \text{and } lst(\tau_i^j) &= min\{(s_i^{j-1} + p_i + \eta_i),\ (s_i^1 + (j-1) \times p_i + (n_i - j + 1) \times \lambda_i)\}. & (7) \end{aligned}$$

If s_i^j is in between the *est*(τ_i^j) and *lst*(τ_i^j), then the estimated *est* and *lst* of $s_i^{n_i}$, based on s_i^j and $s_i^{n_i+1}$, specify a feasible window.

3.2 Cyclic Scheduling Technique

The basic approach of scheduling a set of synchronous periodic tasks is to consider the execution of all instances within the scheduling frame whose length is the LCM of all periods. The release times of the first periods of all tasks are zero. As long as one instance is scheduled in each period within the frame and these executions meet the timing constraints, a feasible schedule is obtained. In a feasible schedule, all instances complete the executions before the LCM.

On the other hand, in asynchronous task systems, as depicted in Figure 2 in which the LCM is $200ms$, the periods of the two tasks are out of phase. It is possible that the completion time of some instance in a feasible schedule exceeds the LCM. To find a feasible schedule for such an asynchronous system, a technique of handling the time value which exceeds the LCM is proposed.

The technique is based on the linked list structure described in the work [CA93]. Without loss of generality, we assume the minimum release time among the first periods of all tasks is zero. We keep a linked list for each processor and a separated list for the communication network. Each element in the list represents a time slot assigned to some instance or communication. The fields of a time slot of some processor p: (1) *task id i* and *instance id j* indicate the identifier of the time slot. (2) *start time st* and *finish time ft* indicate the start time and completion time of τ_i^j respectively. (3) *prev ptr* and *next ptr* are the pointers to the preceding and succeeding time slots respectively. The list is arranged in an increasing order of *start_time*. Any two time slots are nonoverlapping. Since the execution of an instance is nonpreemptable, the time difference between *start_time* and *finish_time* equals the execution time of the task.

- s_i^j is the start time of τ_i^j under σ_m.
- f_i^j is the completion time of τ_i^j under σ_m.
- $r_i^j = p_i \times (j-1) + r_i$, and $d_i^j = p_i \times (j-1) + d_i$.
- $\delta(x) = 0$, if $x \leq 0$; and $= x$, if $x > 0$.
- $\phi(\tau_i)$ is the ID of processor which τ_i is assigned to.
- $\tau_i^j \mapsto t_k^l$ is the communication from τ_i^j to τ_k^l. If $\phi(\tau_i) = \phi(\tau_k)$, then $\tau_i^j \mapsto \tau_k^l$ is a local communication.
- $S(c, \sigma_c)$ is the start time of communication c on the network under σ_c.
- $F(c, \sigma_c)$ is the completion time of communication c on the network under σ_c.

The minimum value of $E(\phi, \sigma_m, \sigma_c)$ is zero. It occurs when the executions of all instances meet the jitter constraints and all communications meet their latency constraints. A feasible multiprocessor schedule can be obtained by collecting the values of s_i^j and f_i^j, $\forall\ i$ and j. Likewise, a feasible network schedule can be obtained from $S(c, \sigma_c)s$ and $F(c, \sigma_c)s$.

Since the task system is asynchronous and the communication pattern could be in the form of cyclic dependency, we solve the problem of finding a feasible solution $(\phi, \sigma_m, \sigma_c)$ by exploiting the cyclic scheduling technique and embedding the technique into the simulated annealing algorithm.

3 The Approach

3.1 Bounds of a Scheduling Window

Define the scheduling window for a task instance as the time interval during which the task can start. Traditionally, the lower and upper bounds of the scheduling window for a task instance are called earliest start time (*est*) and latest start time (*lst*) respectively. These values are given and independent of the start times of the preceding instances.

We consider the scheduling of periodic tasks with relative timing constraints described in Equations 3 and 4. The scheduling window for a task instance is derived from the start times of its preceding instances. A *feasible* scheduling window for a task instance τ_i^j is a scheduling window in which any start time in the window makes the timing relation between s_i^{j-1} and s_i^j satisfy

2.3 Problem Formulation

We consider the static assignment and scheduling in which a task is the finest granularity object of assignment and an instance is the unit of scheduling. We applied the simulated annealing algorithm [KGV83] to solve the problem of real-time periodic task assignment and scheduling with hybrid timing constraints. In order to make the execution of instances satisfy the specifications and meet the timing constraints, we consider a scheduling frame whose length is the least common multiple (LCM) of all periods of tasks. Given a task set Γ and its communications C, we construct a set of task instances, I, and a set of multiple communications, M. We extend each task $\tau_i \in \Gamma$ to n_i instances, τ_i^1, τ_i^2, ..., and $\tau_i^{n_i}$. These n_i instances are added to I. Each communication $\tau_i \mapsto \tau_j \in C$ is extended to $\min(n_i, n_j)$[1] undersampled communications where $n_i = \mathrm{LCM}/p_i$ and $n_j = \mathrm{LCM}/p_j$. These multiple communications are added to M. The extension can be stated as follows.

- If $n_i < n_j$, then $\tau_i \mapsto \tau_j$ is extended to $\tau_i^1 \mapsto \tau_j^?$, $\tau_i^2 \mapsto \tau_j^?$, ..., and $\tau_i^{n_i} \mapsto \tau_j^?$.
- If $n_i > n_j$, then $\tau_i \mapsto \tau_j$ is extended to $\tau_i^? \mapsto \tau_j^1$, $\tau_i^? \mapsto \tau_j^2$, ..., and $\tau_i^? \mapsto \tau_j^{n_j}$.
- If $n_i = n_j$, then $\tau_i \mapsto \tau_j$ is extended to $\tau_i^1 \mapsto \tau_j^1$, $\tau_i^2 \mapsto \tau_j^2$, ..., and $\tau_i^{n_i} \mapsto \tau_j^{n_j}$.

A task ID with a superscript of question mark indicates some instance of the task. For example, $\tau_i^1 \mapsto \tau_j^?$ means that τ_i^1 communicates with some instance of τ_j. We describe how we assign the nearest instance for each communication in Section 4.1.2.

The problem can be formulated as follows. Given a set of task instance, I, its communications M, we find an assignment ϕ, a total ordering σ_m of all instances, and a total ordering σ_c of all communications to minimize

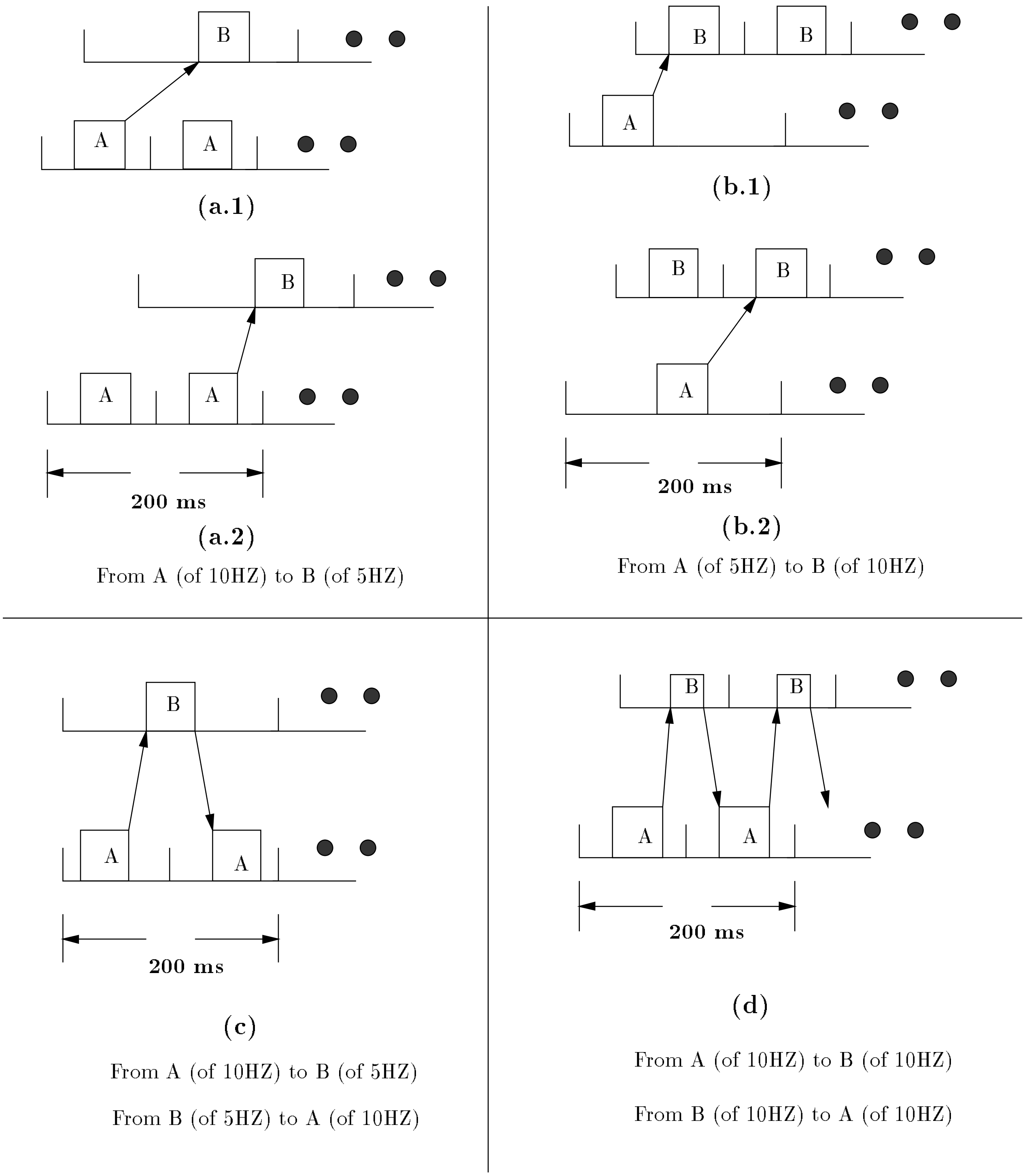
$$\begin{aligned} E(\phi, \sigma_m, \sigma_c) &= \sum_{i,j} \delta(p_i - \lambda_i - s_i^{j+1} + s_i^j) + \sum_{i,j} \delta(s_i^{j+1} - s_i^j - p_i - \eta_i) \\ &+ \sum_{i,j} \delta(f_i^j - d_i^j) + \sum_{i,j,k,l} \delta(F(t_i^j \mapsto t_k^l, \sigma_c) - s_k^l) \\ &+ \sum_{i,j,k,l} \delta(f_k^l - s_i^j - \text{Latency } (\tau_i \text{ to } \tau_k)) \qquad (5) \end{aligned}$$

$$\text{subject to } \; s_i^j \geq r_i^j \text{ and } \; S(t_i^j \mapsto t_k^l, \sigma_c) \geq f_i^j, \;\; \forall \; t_i^j \mapsto t_k^l,$$

where

---

[1] Due to undersampling, when an asynchronous communication is extended to multiple communications, the number of multiple communications is the smaller number of sender and receiver instances.



(a.1)

(a.2)

From A (of 10HZ) to B (of 5HZ)

(b.1)

(b.2)

From A (of 5HZ) to B (of 10HZ)

(c)

From A (of 10HZ) to B (of 5HZ)

From B (of 5HZ) to A (of 10HZ)

(d)

From A (of 10HZ) to B (of 10HZ)

From B (of 10HZ) to A (of 10HZ)

Figure 2: Possible Communication Patterns

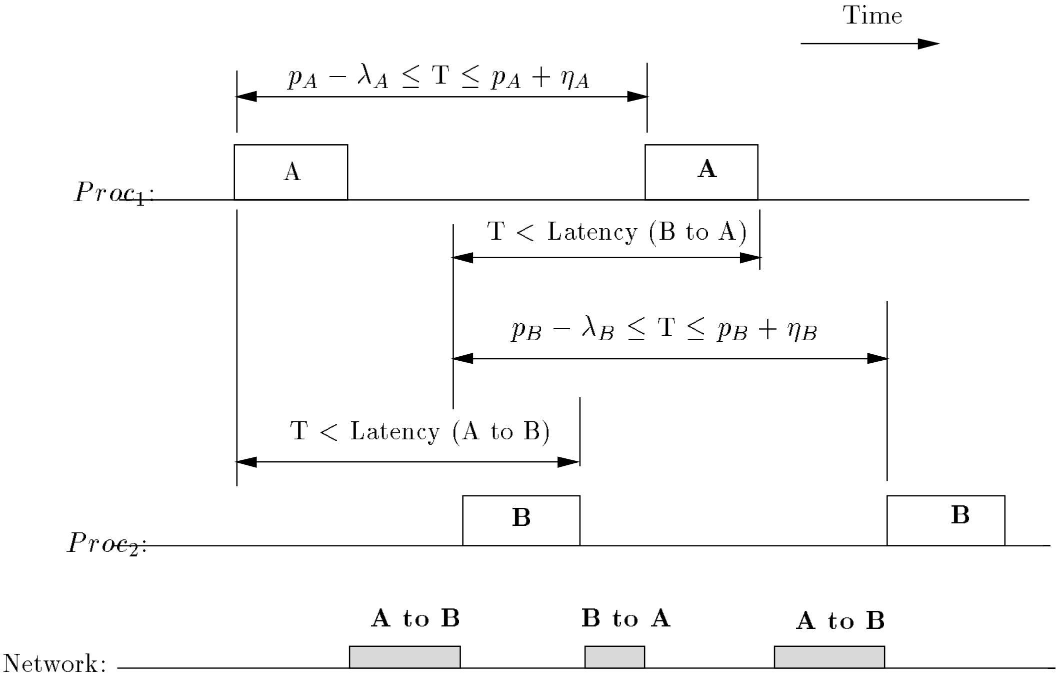


Figure 1: Relative Timing Constraints

frequency is n times of the receiver frequency and no cyclic dependency is involved, then one of every n instances of the sending task has to communicate with one instance of the receiving task. (Examples of this situation are shown in Figures 2.a.1 and 2.a.2. Likewise, for the case in which the receiver frequency is n time that of the sender frequency and no cyclic dependency is present, the patterns are shown in Figures 2.b.1 and 2.b.2. For an asynchronous communication, the sending (receiving) task in low frequency sends (receives) the message to (from) the *nearest* receiving (sending) task as shown in Figure 2.a (2.b). The cases where cyclic dependency is considered are shown in Figures 2.c and 2.d.

2.2 System Model

A real-time DCS consists of a number of processors connected together by a communications network. The execution of an instance on a processor is nonpreemptable. To provide predictable communication and to avoid contention for the communication channel at the run time, we make the following assumptions. (1) Each IPC occurs at the pre-scheduled time as the schedule is generated. (2) At most one communication can occur at any given time on the network.

$$\begin{aligned} s_i^j &\leq s_i^{j-1} + p_i + \eta_i \\ & \forall j = 2, \ldots, n_i + 1. \end{aligned} \tag{4}$$

- **Asynchronous Communication:** Tasks communicate with each others by sending and receiving data or messages. The frequencies of sending and receiving tasks of a communication can be different. In consequence, communications between tasks may cross the task periods. When such asynchronous communications occur, the semantics of undersampling is assumed. When two tasks of different frequencies are communicating, schedule the message only at the lower rate. For example, if task A (of 10HZ) sends a message to task B (of 5HZ), then in every $200ms$, one of two instances of task A has to send a message to one instance of task B. If the sending and receiving tasks are assigned to the same processor, then a local communication occurs. We assume the time taken by a local communication is negligible. When an interprocessor communication (IPC) occurs, the communication must be scheduled on the communications network between the end of the sending task execution and the start of the receiving task execution. The transmission time required to communicate the message i over the network is denoted by μ_i.

- **Communication Latency:** Each communication is associated with a communication latency which specifies the maximum separation between the start time of the sending task and the completion time of the receiving task.

- **Cyclic Dependency:** Research on the allocation problem has usually focused on acyclic task graphs [Ram90, HS92]. Given an acyclic task graph $G = \{V, E\}$, if the edge from task A to task B is in E then the edge from B to A can not be in E. The use of acyclic task graphs excludes the possibility of specifying the cyclic dependency among tasks. For example, consider the following situation in which one instance of task A can not start its execution until it receives data from the last instance of task B. After the instance of task A finished its execution, it sends data to the next instance of task B. Since tasks A and B are periodic, the communication pattern goes on throughout the lifetime of the application. To be able to accommodate this situation, we take cyclic dependency into consideration.

The timing constraints described above are shown in Figure 1. For periodic tasks A and B, the start times of each and every instance of task execution and communication are pre-scheduled such that (1) the execution intervals fall into the range between $p - \lambda$ and $p + \eta$ and (2) the time window between the start time of sending task and the completion time of receiving task is less than the latency of the communication. In Figure 2, we illustrate examples of all possible communication patterns considered in this paper. The description of the communications in the task system is in the form of "*From sender-task-id (of frequency) To receiver-task-id (of frequency)*". If the sender

2 Problem Description

Various kinds of periodic task models have been proposed to represent the real-time system characteristics. One of them is to model an application as an independent set of tasks, in which each task is executed once every period under the ready time and deadline constraints. Synchronization (e.g. precedence and mutual exclusion) and communications are simply ignored. Another model to take the precedence relationship and communications into account is to model the application as a task graph. In a task graph, tasks are represented as nodes while communications and precedence relationship between tasks are represented as edges. The absolute timing constraints can be imposed on the tasks. Tasks have to be allocated and scheduled to meet their ready time and deadline constraints upon the presence of synchronization and communications. The deficiency of task graph modeling is inability of specifying the relative constraints across task periods. For example, one can not specify the minimum separation interval between two consecutive executions of the same task.

In the work [CA93], we modified the real-time system characteristics by taking into account the relative constraints on the instances of a task. We considered the scheduling problem of the periodic tasks with the relative timing constraints. We analyzed the timing constraints and derive the scheduling window for each task instance. Based on the scheduling window, we presented the time-based approach of scheduling a task instance. The task instances are scheduled one by one based on their priorities assigned by the proposed algorithms. In this paper we augment the real-time system characteristics by considering the inter-task communication on DCS.

2.1 Task Characteristics

The problem considered in this chapter has the following characteristics.

- **The Fundamentals:** A task is denoted by the 4-tuple $< p_i, e_i, \lambda_i, \eta_i >$ denoting the period, computation time, low jitter and high jitter respectively. One instance of a task is executed each period. The execution of a task instance is non-preemptable. The start times of two consecutive instances of task τ_i are at least $p_i - \lambda_i$ and at most $p_i + \eta_i$ apart. Let s_i^j and f_i^j be the start time and finish time of task instance τ_i^j respectively. The timing constraints specified in Equations 1 through 4 must be satisfied.

$$f_i^j = s_i^j + e_i \tag{1}$$

$$s_i^{n_i+1} = s_i^1 + \mathrm{LCM} \tag{2}$$

$$s_i^j \geq s_i^{j-1} + p_i - \lambda_i \tag{3}$$

1 Introduction

The task allocation and scheduling problem is one of the basic issues of building real-time applications on a distributed computing system (DCS). DCS is typically modeled as a collection of processors interconnected by a communication network. For hard real-time applications, the allocation of tasks over DCS is to fully utilize the available processors and the scheduling is to meet their timing constraints. Failure to meet the specified timing constraints or inability to respond correctly can result in disastrous consequence.

For the hard real-time applications, such as avionics systems and nuclear power systems, the approach to guarantee the critical timing constraints is to allocate and schedule tasks *a priori*. The essential solution is to find an static allocation in which there exists a feasible schedule for the given task sets. Ramamritham [Ram90] proposes a global view where the purpose of allocation should directly address the schedulability of processors and communication network. A heuristic approach is taken to determine an allocation and find a feasible schedule under the allocation. Tindell *et al.* [TBW92] take the same global view and exploit a simulated annealing technique to allocate periodic tasks. A distributed rate-monotonic scheduling algorithm is implemented. In each period a task must execute once before the specified deadline. The transmission times for the communications are taken into account by subtracting the total communication time from the deadline and making the execution of the task more stringent.

Simply assuring that one instance of each task starts after the ready time and completes before the specified deadline is not enough. Some real-time applications have more complicated timing constraints for the tasks. For example, the relative timing constraints may be imposed upon the consecutive executions of a task in which the scheduling of two consecutive executions of a periodic task must be separated by a minimum execution interval. Communication latency can be specified to make sure that the time difference between the completion of the sending task and the start of the receiving task does not exceed the specified value. The Boeing 777 Aircraft Information Management System is such an example [CDHC94]. For such applications, the algorithms proposed in literature do not work because the timing constraints are imposed across the periods of tasks. In this paper, we consider the relative timing constraints for real examples of real-time applications in Section 2. Based on the task characteristics, we propose the approach to allocate and schedule these applications in Section 3. A simulated annealing algorithm is developed to solve the problem in which the reduction on the search space is given in Section 4. In Section 5, we evaluate the practicality and show the significance of the algorithm. Instead of randomly generating the *ad hoc* test cases, we apply the algorithm to a real example. The example is the Boeing 777 AIMS with various numbers of processors. The experimental results are shown in Section 5.

Allocation and Scheduling of Real-Time Periodic Tasks with Relative Timing Constraints*

Sheng-Tzong Cheng and Ashok K. Agrawala
Institute for Advanced Computer Studies
Systems Design and Analysis Group
Department of Computer Science
University of Maryland
College Park, MD 20742
{stcheng,agrawala}@cs.umd.edu

Abstract

Allocation problem has always been one of the fundamental issues of building the applications in distributed computing systems (DCS). For real-time applications on DCS, the allocation problem should directly address the issues of task and communication scheduling. In this context, the allocation of tasks has to fully utilize the available processors and the scheduling of tasks has to meet the specified timing constraints. Clearly, the execution of tasks under the allocation and schedule has to satisfy the precedence, resources, and other synchronization constraints among them.

Recently, the timing requirements of the real-time systems emerge that the relative timing constraints are imposed on the consecutive executions of each task and the inter-task temporal relationships are specified across task periods. In this paper we consider the allocation and scheduling problem of the periodic tasks with such timing requirements. Given a set of periodic tasks, we consider the least common multiple (LCM) of the task periods. Each task is extended to several instances within the LCM. The scheduling window for each task instance is derived to satisfy the timing constraints. We develop a simulated annealing algorithm as the overall control algorithm. An example problem of the sanitized version of the Boeing 777 Aircraft Information Management System is solved by the algorithm. Experimental results show that the algorithm solves the problem in a reasonable time complexity.

---

*This work is supported in part by Honeywell under N00014-91-C-0195 and Army/Phillips under DASG-60-92-C-0055. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of Honeywell or Army/Phillips.