Resource Scheduling in Dependable Integrated Modular Avionics

Yann-Hang Lee and Daeyoung Kim
CISE Department,
University of Florida
{yhlee, dkim}@cise.ufl.edu

Mohamed Younis, Jeff Zhou, James McElroy
Honeywell International Inc.
{mohamed.younis, jeff.zhou, james.mcelroy}
@honeywell.com

Abstract

In the recent development of avionics systems, Integrated Modular Avionics (IMA) is advocated for next generation architecture that needs integration of mixed-criticality real-time applications. These integrated applications meet their own timing constraints while sharing avionics computer resources. To guarantee timing constraints and dependability of each application, an IMA-based system is equipped with the schemes for spatial and temporal partitioning. We refer the model as SP-RTS (Strongly Partitioned Real-Time System), which deals with processor partitions and communication channels as its basic scheduling entities.

This paper presents a partition and channel-scheduling algorithm for the SP-RTS. The basic idea of the algorithm is to use a two-level hierarchical schedule that activates partitions (or channels) following a distance-constraints guaranteed cyclic schedule and then dispatches tasks (or messages) according to a fixed priority schedule. To enhance schedulability, we devised heuristic algorithms for deadline decomposition and channel combining. The simulation results show the schedulability analysis of the two-level scheduling algorithm and the beneficial characteristics of the proposed deadline decomposition and channel combining algorithms.

1. Introduction

Advances in computer and communication technology have introduced new architectures for avionics systems, which emphasize the integration of applications, dependability, and cost reduction. Away from the traditional federated implementation for avionics systems, the new approach, referred to as Integrated Modular Avionics (IMA) [1], utilizes multiple standardized processor modules in building functional components of avionics systems. It allows the applications to be merged into an integrated system. While permitting resource sharing, the approach employs temporal and spatial partitioning to set up the application boundaries needed to maintain system predictability, real-time response, and dependability [2, 6]. For the interactions between applications, it adopts a message model that can easily accommodate replicated executions of mission-critical applications.

Under the IMA architecture, each processor can host multiple partitions in which applications can be executed using the assigned resources. Spatial partitioning implies that a partition cannot access other partition's resources, like memory, buffers, and registers. On the other hand, temporal partitioning guarantees a partition's monopoly use of a pre-allocated processing time without any intervention from other partitions. Thus, a partition is the sole owner of its resources, such as memory segments, I/O devices, and processor time slots. As a result, the applications running in different partitions cannot interfere with each other. To facilitate communications between applications, each partition can be assigned with one or more communication channels. An application can transmit messages during the slots allocated to its channel and access exclusively the channel buffers. In this sense, the channels are spatial and temporal partitions of communication resource and are dedicated to one message-sending application.

An application running within a partition can be with multiple cooperating tasks. For instance, the Honeywell's Enhanced Ground Proximity Warning System (EGPWS) consists of tasks for map loading, terrain threat detection, alert prioritization, display processing, etc. With the spatial and temporal partitioning, the EGPWS application can be developed separately and then integrated with other applications running in different partitions of an IMA-based system. Its execution cannot be affected by any malfunctions of other applications (presumably developed by other manufactures) via wild writes or task overruns. However, sufficient resources must be allocated to the partition and the channels, so that the EGPWS application can ensure a proper execution and meet its real-time constraints.

One apparent advantage of IMA-based systems with spatial and temporal partitioning is that each application is running in its own environment. Thus, as long as the partition environment is not changed, an application's behavior remains constant even if other applications are modified. This leads to a crucial advantage to avionics systems, i.e. when one application is revised, other applications don't need to be re-certified by the FAA. Thus, the integration of applications in a complex system can be upgraded and maintained easily. It is conceivable that such architecture with spatial and temporal partitioning can be useful for integrating general real-time applications, and will be referred to as a strongly partitioned real-time system (SP-RTS) in the paper.

In this paper, we investigate the issues related to the partition and channel scheduling in SP-RTS. To schedule processor execution, we need to determine which partition is active and to select a task from the active partition for execution. According to temporal partitioning, time slots are allocated to partitions. Within each partition, fixed priorities are assigned to tasks based on rate-monotonic or deadline-monotonic algorithms [14, 5]. A lower priority task can be preempted by higher priority tasks of the same partition. In other words, the scheduling approach is hierarchical that partitions are scheduled following a cyclic schedule and tasks are dispatched according to a fixed priority schedule. We can conjecture a real system where partitions are processes with protected memory spaces and tasks are threads in a process. At process level, a cyclic scheduling is employed, whereas, in thread level, thread priorities are compared. The scheme doesn't need to make a global priority comparison between threads of different processes. Similar hierarchical scheduling is also applied to the communication media where channels are scheduled in a cyclic fashion and have enough bandwidth to guarantee message communication. Within each channel, messages are then ordered according to their priorities for transmission.

Given task execution characteristics, we are to determine the cyclic schedules for partitions and channels under which the computation results can be delivered before or on the task deadlines. The problem differs from the typical cyclic scheduling since, at the partition and channel levels, we don't evaluate the invocations for each individual task or message. Only aggregated task execution and message transmission models are considered. In addition, the scheduling for partitions and channels must be done collectively such that tasks can complete their computation and then send out the results without missing any deadlines.

A different two-level hierarchical scheduling scheme has been proposed by Deng and Liu in [8]. The scheme allows real-time applications to share resources in an open environment. The scheduling structure has an earliest-deadline-first (EDF) scheduling at the operating system level. The second level scheduling within each application can be either time-driven or priority-driven. For acceptance test and admission of a new application, the scheme analyzes the application schedulability at a slow processor. Then, the server size is determined and server deadline of the job at the head of the ready queue is set at run-time. Since the scheme does not rely on fixed allocation of processor time or fine-grain time slicing, it can support various types of applications, such as release time jitters, non-predictable scheduling instances, and stringent timing requirements.

The scheduling approach for avionics applications under the APEX interface of IMA architecture was discussed by Audsley and Wellings [4]. A recurrent solution to analyze task response time in an application domain is derived and the evaluation results show that there is a potential for a large amount of release jitter. However, the paper does not address the issues of constructing cyclic schedules at the operating system level. To remedy the problem, our first step is to establish scheduling requirements for the cyclic schedules such that task schedulability under a given fixed priority schedules within each partition can be ensured. The approach we adopt is similar to the one in [8] of comparing the task execution in SP-RTS environment with that at a dedicated processor. The cyclic schedule then tries to allocate partition execution intervals by "stealing" task inactivity periods. This stealing approach resembles the slack stealer for scheduling soft-aperiodic tasks in fixed priority systems [11]. Once the schedulability requirements are obtained, suitable cyclic schedules can be constructed. Following the partitioning concept of IMA, the operating system level cyclic schedule is flexible to support system upgrade and integration. It is designed in a way that no complete revision of scheduling algorithms is required when the workload or application tasks in one partition are modified.

The rest of the paper is organized as follows. In section 2, we describe the system models that describe tasks, partition servers, messages, and channel servers in SP-RTS. Then, we show the overall system scheduling algorithm and its specific components, such as deadline decomposition, task and message schedulability checking, channel combining, and cyclic scheduling for partition servers and channel servers in section 3. Evaluation results are presented in section 4. A conclusion is then given in section 5.

2. *System Models*

The SP-RTS system model, as shown in Figure 1, includes multiple processors inter-connected by a time division multiplexing communication bus such as ARINC 659 [3]. Each processor has several execution partitions to which applications can be allocated. An application consists of multiple concurrent tasks that can communicate with each other within the application partition. Task execution is subject to deadlines. Each task must complete its computation and send out the result messages on time in order to meet its timing constraints.

Messages are the only form of communication among applications, regardless of whether their execution partitions are in the same processor or not. For inter-partition communication, the bandwidth of the shared communication media is distributed among all applications by assigning channels to a subset of tasks running in a partition. We assume that there are hardware mechanisms to enforce the partition environment and channel usage by each application, and to prevent any unauthorized accesses. Thus, task computation and message transmission are protected in their application domain. The mechanisms could include memory protection controller, slot/channel mapping, and separate channel buffers.

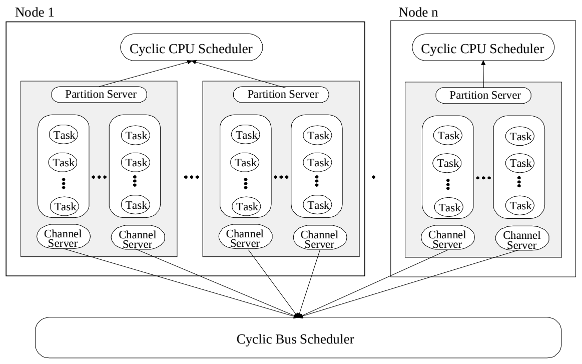


Figure 1. The architecture model for strongly partitioned real-time systems (SP-RTS)

In our task model, we assume that each task arrives periodically and needs to send an output message after its computation. Thus, as illustrated in Figure 2, tasks are specified by several parameters, including invocation period (T_i), worst-case execution time (C_i), deadline (D_i) and message size (M_i). Note that, to model sporadic tasks, we can assign the parameter T_i as the minimum inter-arrival interval between two consecutive invocations.

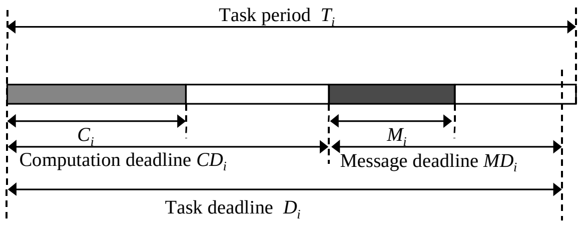


Figure 2. Task model and deadlines

In order to schedule tasks and messages at processors and communication channels, the task deadline, D_i, is decomposed into message deadline (MD_i) and computation deadline (CD_i). The assignment of message deadlines influences the bandwidth allocation for the message. For example, when the message size, M_i, is 1K slots, and the message deadline of 10*ms*, then the bandwidth requirement is 0.1M slots per second. In the case of the 1*ms* message deadline, the bandwidth requirement becomes 1M slots per second. However, a tradeoff must be made since a long message deadline implies a less amount of bandwidth to be allocated, thus the task computation has to be completed immediately.

For each processor in SP-RTS architecture, the scheduling is done in a two-level hierarchy. The first level is within each partition server where the application tasks are running and a higher priority task can preempt any lower priority tasks of the same partition. The second level is a cyclic partition schedule that allocates execution time to partition servers of the processor. In other word, each partition server, S_k, is scheduled periodically with a fixed period. We denote this period as the *partition cycle*, η_k. For each partition cycle, the server can execute the tasks in the partition for an interval $\alpha_k\eta_k$ where α_k is less than or equal to 1 and is called *partition capacity*. For the remaining interval of $(1-\alpha_k)\eta_k$, the server is blocked. In Figure 3, an example execution sequence of a partition that consists of three tasks is depicted. During each partition cycle, η_k, the tasks, τ_1, τ_2, and τ_3, are scheduled to be executed for a period of $\alpha_k\eta_k$. If there is no active task in the partition, the processor is idle and cannot run any active tasks from other partitions.

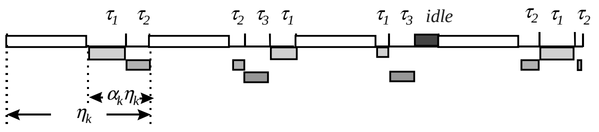
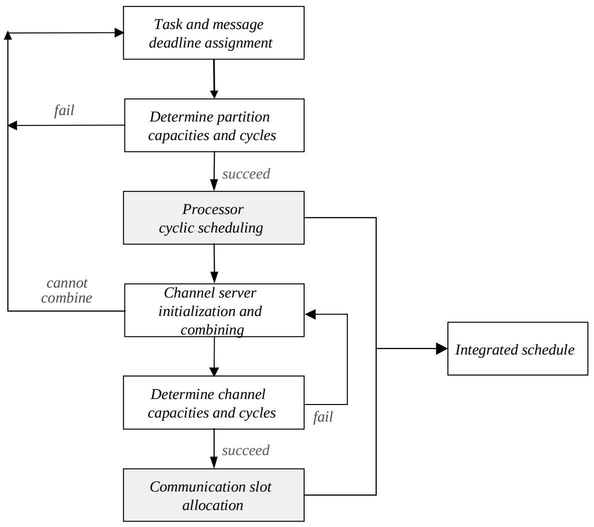


Figure 3. An illustrative task and partition execution sequence

Similarly, a two-level hierarchical scheduling method is applied to the message and channel scheduling. A channel server provides fixed-priority preemptive scheduling for messages. Then, a cyclic schedule assigns a sequence of communication slots to each channel server according to its channel cycle, μ_k, and channel capacity, β_k. A channel may send out messages using $\beta_k\mu_k$ slots during every period of μ_k slots. Note that we use the unit of "slot" to indicate both message length and transmission time, with an assumption that communication bandwidth and slot length are given. For instance, a 64-bit slot in the 30MHz 2-bit wide ARINC 659 bus [3] is equivalent to 1.0667μs, and a message of 1000 bytes will be transmitted in 125 slots. For convenience purposes, we define the conversion factors *ST* as a slot-to-time ratio based on slot length and bus bandwidth.

3. Scheduling Approach

The objective of our scheduling approach is to find feasible cyclic schedules for partition and channel servers which process tasks and transmit messages according to their fixed priorities within the servers. With proper capacity allocation and frequent invocation at each server, the combined delays of task execution and message transmission are bounded by the task deadlines. In Figure 4, we show the overall approach which first applies a heuristic deadline decomposition to divide the problem into two parts: partition-scheduling and channel-scheduling. If either one cannot be done successfully, the approach iterates with a modified deadline assignment. We also assume that the initial task set imposes a processor utilization and a bus utilization less than 100% and each task's deadline is larger than its execution time plus its message transmission time, i.e., $D_i \geq C_i + ST*M_i$ for task i.



Task and message deadline assignment
Determine partition capacities and cycles
fail
succeed
Processor cyclic scheduling
cannot combine
Channel server initialization and combining
Integrated schedule
Determine channel capacities and cycles
fail
succeed
Communication slot allocation

Figure 4. Combined partition and channel scheduling approach

3.1. Deadline Decomposition

It is necessary to decompose the original task deadline, D_i, into computation and message deadline, CD_i and MD_i, for every task, before we can schedule the servers for partition execution and message transmission. A deadline decomposition algorithm is used to assign these deadlines in a heuristic way. If we assign tight message deadlines, messages may not be schedulable. Similarly, if tasks have tight deadlines, processor scheduling can fail. The following equation is used to calculate the message deadline and computation deadline for each task:

Message Deadline, $MD_i = (D_i \dfrac{ST * M_i}{C_i + ST * M_i}) f_i$

Computation Deadline, $CD_i = D_i - MD_i$

where f_i is an adjusting factor for each task. The main idea of deadline decomposition is that it allocates the deadlines, CD_i and MD_i, proportionally to their time requirements needed for task execution and message transmission. In addition, the adjusting factor f_i is used to calibrate the computation and message deadlines based on the result of previous scheduling attempts and the utilization at processor and communication bus. Since the message and task deadlines must be lower-bounded to the transmission time ($ST*M_i$) and computation time (C_i), respectively, and upper-bounded to D_i, we can obtain the lower bound and upper bound of the adjusting factor f as

$$\frac{1}{D_i(\frac{1}{C_i + ST * M_i})} \leq f_i \leq \frac{D_i - C_i}{D_i(\frac{ST * M_i}{C_i + ST * M_i})}$$

Since an adjusting factor of 1.0 is a fair distribution and always included in the range of f_i, we set the initial value of f_i to be 1. The heuristic deadline decomposition, as show in Figure 5, is similar to a binary search algorithm in the attempt of finding the right proportion of task and message deadlines. If we reach the situation that it cannot assign new value for all tasks, we declare the input set of tasks as unschedulable.

```
Initialization for all tasks
    MinF = 1 / (D_i * (1/(C_k+ ST*M_k)));
    MaxF = (D_i-C_i) / (D_i * (ST*M_i /(C_i+ ST*M_i)));
    f_i = 1.0;

Iterative change of f_k when either partition or channel
scheduling fails
    If (Partition scheduling fails) {
        MaxF = f_i; f_i = (MinF + f_i) / 2.0;
    }
    else if (Channel scheduling fails) {
        MinF = f_i; f_i = (MaxF + f_i) / 2.0;
    }
```

Figure 5. The deadline decomposition algorithm

3.2. Partition and Channel Scheduling

In SP-RTS, partitions and channels are cyclically scheduled. The partition cyclic schedule is based on partition cycle, η_k, and partition capacity, α_k. Similarly, a channel cyclic schedule with parameters, β_k and μ_k implies that the channel can utilize $\beta_k\mu_k$ slots during a period of μ_k slot interval. While tasks and messages are scheduled according to their priority within the periodic servers, the cyclic schedule determines the response time of task execution and message transmission. In this subsection, we give a short description of the scheduling theory that can be used to schedule the cyclic partition and channel servers. A full discussion of the scheduling theory and the associated proof are given in our previous paper [10].

Note that, at the system level, the partition server S_k is cyclically scheduled with a fixed partition cycle, η_k. For every partition cycle, the server can execute the task in partition P_k during an interval of $\alpha_k\eta_k$ where $\alpha_k \le 1$. For the remaining interval of $(1-\alpha_k)\eta_k$, the server is blocked. Suppose that there are n tasks in partition server S_k listed in priority order such that $\tau_1 < \tau_2 < \tau_3 < \ldots < \tau_n$ where τ_1 has the highest priority and τ_n the lowest. According to deadline monotonic algorithm, we assume that the highest priority is given to the task with shortest task deadline. In order to evaluate the schedulability of the partition server, S_k, we first consider that the task set is executed at a dedicated processor of capacity α_k. Based on the necessary and sufficient condition of schedulability analysis [12, 13], task τ_i is schedulable if there exists a t 0 $H_i = \{CD_i \cup lT_j \mid j=1,2,\ldots,i-1;\ l=1,2,\ldots, \lfloor CD_i/T_j \rfloor\}$such that:

$$W_i(\alpha_k, t) = \sum_{j=1}^{i} \frac{C_j}{\alpha_k} \left\lceil \frac{t}{T_j} \right\rceil \le t$$

The expression $W_i(\alpha_k, t)$ indicates the worst cumulative execution time demand on the processor made by the tasks with a priority higher than or equal to τ_i during the interval $[0,t]$. We now define $B_i(\alpha_k) = \max_{t \in H_i} \{t - W_i(\alpha_k, t)\}$ and $B_0(\alpha_k) = \min_{i=1,2,..n} B_i(\alpha_k)$, where n is the total number of tasks in the partition. Note that, when τ_i is schedulable, $B_i(\alpha_k)$ represent the total period in the interval $[0, t]$ that the processor is not running any tasks with a priority higher than or equal to that of τ_i in the partition server. $B_i(\alpha_k)$ is equivalent to the level-i inactivity period in the interval $[0, t]$ [11].

By comparing the task executions at server S_k and at a dedicated processor of capacity α_k, we can obtain the following theorem [10].

***Theorem 1**. The partition server S_k is schedulable if S_k is schedulable at a dedicated processor of capacity α_k, and $\eta_k \le B_0(\alpha_k)/(1-\alpha_k)$*

Note that $B_0(\alpha_k)$ is a non-decreasing function of α_k. There is a minimum α_k such that $B_0(\alpha_k)$ equals to zero, i.e., a zero inactive period for at least one task in the partition. The minimum α_k indicates the minimum processor capacity needed to schedule the partition. Thus, partition scheduling can fail if the sum of the minimum α_k, for all partitions in a processor, is larger than 1.

With Theorem 1, we can depict the plot of maximum partition cycle vs. the assigned capacity α_k. To illustrate the result, we consider an example in Table 1 in which four application partitions are allocated in a processor. Each partition consists of several periodic tasks and the corresponding parameters of (C_i, T_i) are listed in the Table. Tasks are set to have deadlines equal to their periods and are scheduled within each partition according to a rate-monotonic algorithm. The processor utilization demanded by the 4 partitions, ρ_k, are 0.25, 0.15, 0.27, and 0.03, respectively.

Table 1. Task parameters for the example partitions

	Partition 1 (utilization=0.25)	Partition 2 (utilization=0.15)
tasks (C_i, T_i)	(4, 100) (9, 120) (7, 150) (15, 250) (10, 320)	(2, 50) (1, 70) (8, 110) (4, 150)
	Partition 3 (utilization=0.27)	Partition 4 (utilization=0.03)
tasks (C_i, T_i)	(7,80) (9,100) (16,170)	(1,80) (2,120)

In Figure 6, the curves $\eta_k = B_0(\alpha_k)/(1-\alpha_k)$ are plotted for the example 4 partitions. If the points below the curves are chosen to set up cyclic scheduling parameters for each partition, the tasks in the partition are guaranteed to meet their deadlines.

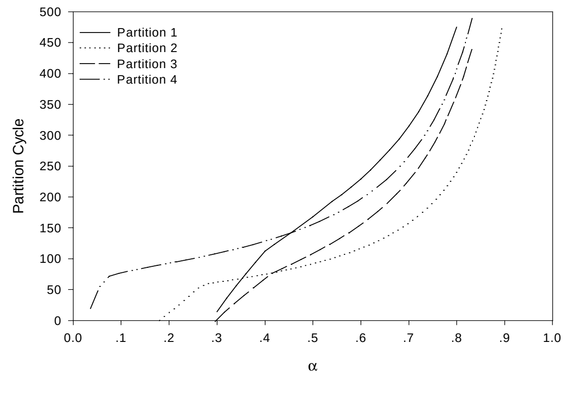


Figure 6. Partition Cycles vs. Processor Capacities for the Example Partitions

For instance, the curve for partition 2 indicates that, if the partition receives 28% of processor capacity, then its tasks are schedulable as long as its partition cycle is less than or equal to 59 time units. Note that the maximum partition cycles increase as we assign more capacity to each partition. This increase is governed by the accumulation of inactivity period when α_k is small. Then, the growth follows by a factor of $1/(1-\alpha_k)$ for a larger α_k. The curves in Figure 6 show that there are sharp rises of the maximum partition cycle when we increase α_k just beyond the minimum required capacities. The rises indicate that a small amount of extra capacity can enlarge the inactive period of a partition server significantly.

According to the design objectives, there are several methods we can use to choose a set of (α_k, η_k) for all partition servers. For instance, we can calculate the minimum α_k first. If the sum of the minimum α_k, for all partition server S_k, and the reserved portion of processor capacity, is less than 100%, the extra capacity can be allocated to all partitions proportionally to their minimum α_k. Then, η_k can be calculated based on Theorem 1. The other approach is to search for the saddle point in the $B_0(\alpha_k)/(1-\alpha_k)$ curve where the initial rise just begins to slow down. The pair (α_k, η_k) at the saddle point is used as the initial capacity allocation and partition cycle. Further increase or reduction can be done proportionally if the total capacity allocated is less than or larger than 1.

We can use the same scheduling method of the partition scheduling for channel scheduling. A channel server, G_k, transmits its messages according to a fixed priority preemptive scheduling method. It provides a bandwidth of $\beta_k\mu_k$ slots to the messages in the channel during every channel cycle, μ_k, where $\beta_k \leq 1$. For the remaining slots of $(1-\beta_k)\mu_k$, the channel server is blocked. Since each channel server follows the identical two-level hierarchical scheduling as partition servers, Theorem 1 can directly applied to obtain the pair of parameters (β_k, μ_k). However, there are several differences. First, only integer number of slots can be assigned to a channel server. Thus, we can use either $\lceil \beta_k\mu_k \rceil$ slots or restrict $\beta_k\mu_k$ to be integer. The second difference is that the message arrivals are not always periodic due to possible release jitters. Release jitters can be included in the schedulability test if they are bounded by some maximum value [15]. The release jitter can also be eliminated if the communication controller incorporates a timed message service that becomes active immediately after the computation deadline is expired. The last difference is the assignment of messages into a channel. According to the principle of partitioning, tasks from different partitions cannot share the same channel for message transmission. For the tasks in a partition, we can group a subset of tasks and let them share a channel server. The grouping can be done based on the semantics of the messages or other engineering constraints. Also, the multiplexing of messages in a shared channel may lead to a saving of bandwidth reservation. We should address this issue in the following subsection.

3.3. Channel Combining

For a channel server that transmits a periodic message with a deadline MD_i and a message size M_i, we must allocate a minimum bandwidth of M_i/MD_i. Since there is a limitation in the total bus bandwidth, we may not always assign one channel server to each message. However, we may be able to combine some messages and let them share a common channel server. This can lead to a bandwidth reduction since the reserved bandwidth can be better utilized by the messages of different deadlines. For example, given two messages 1 and 2 with parameters (M_1, MD_1, T_1) and (M_2, MD_2, T_2), respectively, the minimum bandwidth requirements, in terms of slots per time unit, for separate channels of messages 1 and 2, and for the combined channel, can be computed as following:

$$CB_1 = M_1/MD_1, \quad CB_2 = M_2/MD_2,$$
$$CB_{12} = \max\{ M_1/MD_1, (M_2+M_1 * \lceil MD_2 / T_1 \rceil)/MD_2 \}$$

We assume that message 1 has a higher priority than message 2 in the above computation. The cost of message preemption is ignored which can be at most one slot per preemption since we assume that slots are the basic transmission units in the communication bus. Notice that CB_{12} is not always less that CB_1+CB_2. However, if message 1 has a much shorter deadline comparing with its period and message 2 has a longer deadline than message 1's period, then the bandwidth reduction $CB_1+CB_2-CB_{12}$ becomes substantial. While we reserve a proper amount of bandwidth for an urgent message, the channel is only partially utilized if the message arrives infrequently. This provides a good chance to accommodate additional messages in the same channel and results in a reduction in the required bandwidth.

The above equation also implies that the maximum bandwidth reduction can be obtained by combining the message with a long deadline and the message with a short deadline where the period of the latter should be greater than message deadline of the former. With this observation, we devise a heuristic channel-combining algorithm which is shown in Figure 7. The computation of the minimum bandwidth requirement of a channel consisting of messages $1,2,\ldots,k-1$, and k, is:

$$CB_{12\ldots k} = \max_{j=1,k}\{((\sum_{i=1}^{j-1} M_i * \left\lceil \frac{MD_j}{T_i} \right\rceil + M_j)/MD_j)\}$$

where we assume that message j has a higher priority then message $j+1$. Note that the real bandwidth allocation must be determined according to the choice of channel cycle as described in Theorem 1. However, in order to calculate channel cycle and capacity, the messages in each channel must be known. The channel-combining algorithm outlined in Figure 7 is developed to allocate messages to channels for each partition and to reduce the minimum bandwidth requirement to a specific threshold. If the combined channels cannot be scheduled, we can further decrease the target threshold until no additional combining can be done.

```
Initialization (Channel combining is allowed to the tasks in the same partition)

    Assign one channel server $G_k$ to the message of each task

Iterate the following steps until the sum of total $CB_k$ is less than the target threshold

    determine all pair of combinable channel server $G_k$ and $G_j$ where the max. message deadline in $G_k$ is larger than the min. task period in $G_j$

    For every pair of combinable channel servers $G_k$ and $G_j$ {
        calculate the bandwidth reduction $CB_k+CB_j-CB_{kj}$
    }

    Combine $G_j$ with the server $G_k$ that results in the maximum reduction
```

Figure 7. A heuristic channel combining algorithm

3.4. Cyclic Scheduling for Partition and Channel Servers

Let a feasible set of partition capacities and cycles be $(\alpha_1, \eta_1), (\alpha_2, \eta_2), \dots, (\alpha_n, \eta_n)$ and the set be sorted in the non-decreasing order of η_k. The set cannot be directly used in a cyclic schedule that guarantees the distance constraint of assigning α_k processor capacity for every η_k period in a partition. To satisfy the distance constraint between any two consecutive invocations, we can adopt the pinwheel scheduling approach [7, 9] and transfer $\{\eta_k\}$ into a harmonic set through a specialization operation. Note that, in [9], a fixed amount of processing time is allocated to each task and would not be reduced even if we invoke the task more frequently. This can lead to a lower utilization after the specialization operations. For our partition-scheduling problem, we allocate a certain percentage of processor capacity to each partition. When the set of partition cycles $\{\eta_k\}$ is transformed in to a harmonic set $\{h_k\}$, this percentage doesn't change. Thus, we can schedule any feasible sets of (α_k, η_k) as long as the total sum of α_k is less than 1.

A simple solution for a harmonic set $\{h_k\}$ is to assign $h_k=\eta_1$ for all k. However, since it chooses a minimal invocation period for every partition, a substantial number of context switches between partitions could occur. A practical approach of avoiding excessive context switches is to use Han's S_X specialization algorithm with a base 2 [9]. Given a base partition cycle η, the algorithm finds a h_i for each η_i that satisfies:

$$h_i = \eta * 2^j \le \eta_i < \eta * 2^{j+1} = 2*h_i,$$

To find the optimal base η in the sense of processor utilization, we can test all candidates η in the range of $(\eta_1/2,\ \eta_1]$ and compute the total capacity $\sum_k \alpha_k$. To obtain the total capacity, the set of η_k is transferred to the set of h_k based on corresponding η and then the least capacity requirement, α_k^h, for partition cycle h_k is obtained from Theorem 1. The optimal η is selected in order to minimize the total capacity. In Figure 8, we show a fixed cyclic processor scheduling example that guarantees distance constraint for the set of partition capacities and cycles, *A(0.1,12), B(0.2,14), C(0.1,21), D(0.2,25), E(0.1,48),* and *F(0.3,50)*. We use the optimal base of 10 to convert the partition cycles to *10, 10, 20, 20, 40,* and *40,* respectively.

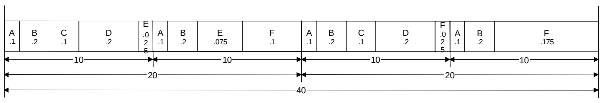
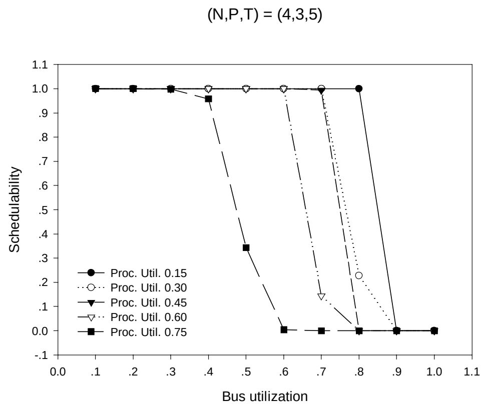


Figure 8. Example of processor cyclic scheduling

The basic method of cyclic scheduling for channel servers is same as that of partition server scheduling. The only difference is that we need to consider that channel bandwidth allocation must be done based on integer number of slots. Let the feasible bus bandwidth capacity allocation set be $(\beta_1, \mu_1), (\beta_2, \mu_2), \dots, (\beta_n, \mu_n)$. Using the S_X specialization, the set $\{\mu_k\}$ will be transformed to a harmonic set $\{m_k\}$. Then, based on Theorem 1 and the reduced m_k, we can adjust the channel capacity β_k to β_k^h subject to $\sum_{i=1}^{n} \lceil \beta_i^h m_k \rceil \le m_k$. There will be $\lceil \beta_k^h\ m_k \rceil$ slots allocated to the channel server G_k.

4. Algorithm Evaluation

In this section, we present the evaluation results of the proposed algorithms for SP-RTS. First, we show the percentage of schedulable task sets in terms of processor and bus utilization under the two-level scheduling, deadline decomposition and channel combining algorithms. Then, we show that the penalty of the



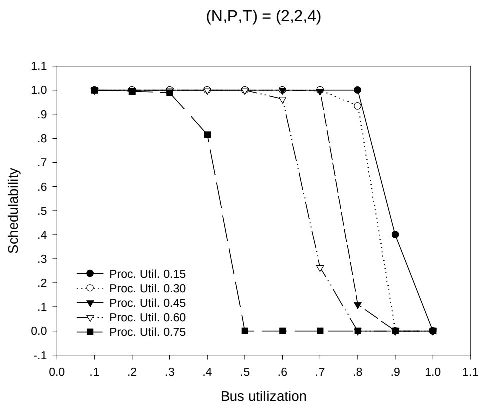


Figure 9. Schedulability test for configurations (4, 3, 5) and (2, 2, 4)

harmonic transformation even if channel server scheduling is negligibly small. Finally, the characteristic behavior of deadline decomposition is illustrated. The evaluations are done with random task and message sets that are generated with specific processor and bus utilization.

4.1. Schedulability Test

A schedulability test of the algorithm is obtained using the simulations of a system model that composes of four processors, three partitions per each processor and five tasks per each partition, i.e., a configuration of (4, 3, 5). The simulations use random task sets that result in variable processor utilization of 15%, 30%, 45%, 60% and 75%. The task periods are uniformly distributed between the minimum and maximum periods. The total processor utilization is randomly distributed to all tasks in each processor and is used to compute the task execution times. To create message sets, we vary the total bus utilization from 10% to 90%. Message lengths are computed with a random distribution of the total bus utilization and task periods.

Using the scheduling procedure of Figure 4, we first assign task and message deadlines for each task. Then the partition capacity and cycle for each partition are computed and the cyclic schedule for each processor is constructed. To schedule message transmission, messages are combined into channels in order to reduce bandwidth requirement. After channel cycle and capacity are determined, a cyclic schedule is formed. For the priority schedules within partitions and channels, we adopt the deadline monotonic approach to order the task and message priorities. With all randomly created task sets, we report the percentage of schedulable task sets among all sets in Figure 9. The figure shows the algorithms are capable of finding proper deadline assignments and, then, determining feasible partition and channel cyclic schedules. For instance, consider the case of 60% processor and bus utilization. Even if the deadlines are less than task periods, almost 100% of task sets are schedulable. Figure 9 also reports the test results of the configuration (2, 2, 4). The curves have the similar trends as that of the configuration of (4, 3, 5).

4.2. The Effects of Deadline Decomposition and Channel Combining Algorithm

It is worthy to look into how the bus is utilized in the channel schedules resulted from the heuristic algorithms of deadline decomposition and channel combining. Consider the following measures:

1. *Measure1* is the bus utilization which equals to the sum of $(ST*M_i)/T_i$ for all tasks. No real-time constraint of message delivery is considered in this measure.
2. *Measure2* is the total bus capacity needed to transmit messages on time with no channel combining (i.e., each task has a dedicated channel). This capacity will be equal to the summation of $(ST*M_i)/MD_i$ for all tasks and can be computed after message deadlines are assigned.
3. *Measure3* is the minimum bus capacity needed to schedule channels. This measure is equal to the summation of minimum β_k for all channels. Note that, according to Theorem 1, the minimum β_k for a channel is defined as the minimum capacity that results in a zero inactive period for

at least one message in the channel. It can be determined after message deadlines are assigned and messages are combined into the channel.

4. *Measure4* is the total bus capacity selected according to Theorem 1. This measure can be formulated as the summation of β_k for all channels.
5. *Measure5* is the final bus capacity allocated to all channels based on a harmonic set of channel cycles and the integer number of slots for each channel. The capacity is equal to the summation of $\lceil \beta_k^h m_k \rceil / m_k$ for all channels.

We can expect an order of *Measure2> Measure5> Measure4> Measure3> Measure1* among the measures. *Measure2* should be much higher than other measures as we allocate bandwidth for each message independently to ensure on schedule message delivery. With the message multiplexing within each channel, the on schedule message delivery can be achieved with a less amount of bandwidth. However, a bandwidth allocation following *Measure3* cannot be practical since the channel cycles must be infinitely small. According to Theorem 1, *Measure4* contains additional capacity that is added to each channel to allow temporary blocking of message transmission during each channel cycle. Furthermore, in *Measure5*, an extra capacity is allocated as we make integer number of slots for each channel and construct a cyclic schedule with harmonic periods.

The simulation results of the above measures are shown in Figure 10. The results confirm our expectation of the order relationship. However, when we change the bus utilization from 0.1 to 0.8, the curves are not monotonically increasing (except the curve of *Measure1*). This is the consequence of the deadline decomposition (*DD*) algorithm. When channels don't have enough bandwidth to meet short message deadlines, the algorithm adjusts the factor f_k and assigns longer deadlines for message transmission. As shown in Figure 5, the *DD* algorithm uses an approach similar to binary search algorithm and makes a big increase to f_k initially. This results in long deadlines and the reduced capacity allocations in *Measure2-5*. In fact, when the bus utilization is less than 30%, the average number of iterations performed in the *DD* algorithm is slightly larger than 1, i.e., only the initial f_k is used to allocate deadlines. When the bus utilization is raised to 40% to 70%, the average number of iterations jumps to 1.6, 1.98, 2.0, and 2.04, respectively. It further increases to 11.09 when the bus utilization is set to 80%.

Figure 10 also illustrates the magnitude of the measures and the differences among them. The gap between *Measure3* and *Measure2* is very visible. This difference is the product of channel combining algorithm. In order to meet a tight message deadline, we have to reserve a large amount of bandwidth. With channel combining, messages of different deadlines share the allocated slots. As long as the message with a shorter deadline can preempt the on-going transmission, the slots in each channel can be fully utilized by multiplexing and prioritizing message transmissions. There is a moderate gap between *Measure3* and *Measure4*. As indicated in Theorem 1, we search for a channel capacity and a channel cycle located in the knee of the curve $\eta_k \le B_0(\alpha_k)/(1-\alpha_k)$ after the initial sharp rise. This implies that a small increase of β_k will be added to *Measure3* in order to obtain a reasonable size of channel cycle. Finally, the difference between *Measure4* and *Measure5* is not significant at all. It is caused by the process of converting η_k to a harmonic cycle m_k, and by allocating an integer number of slots $\lceil \beta_k^h m_k \rceil$ for each channel.

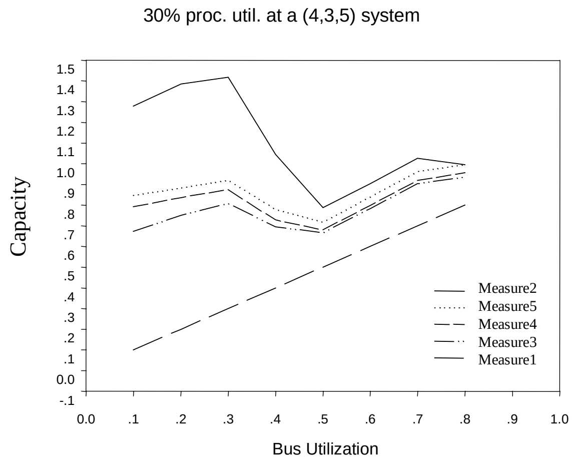


Figure 10. Measures for Bus Utilization and Capacities

The other way of looking into the behavior of the deadline decomposition algorithm is to investigate the resultant decomposition of task deadline, D_i. In Figure 11, we showed the average ratio of message deadline to task deadline, under different processor and bus utilization. If the adjustment factor f_i is constant, the ratio,

$$\frac{MD_i}{D_i} = \left(\frac{ST * M_i}{C_i + ST * M_i}\right) f_i ,$$

should follows a concave curve as we increase bus utilization (by increasing message length, M_i). For instance, when the processor utilization is 15%, there are two segments of concave curves from bus utilization 10% to 70% and from 70% to 90%. The segmentation indicates a jump in the adjustment factors resulted from the deadline decomposition algorithm. In Figure 11, the concavity and the segmentation can also be seen in other curves that represent the message deadline ratios of different processor utilization. When the processor utilization is high, f_i may be modified gradually and partition scheduling may fail if we introduce a sharp increase to f_i. Thus, the concavity and the segmentation are not so obvious as the deadline ratio in an underutilized processor.

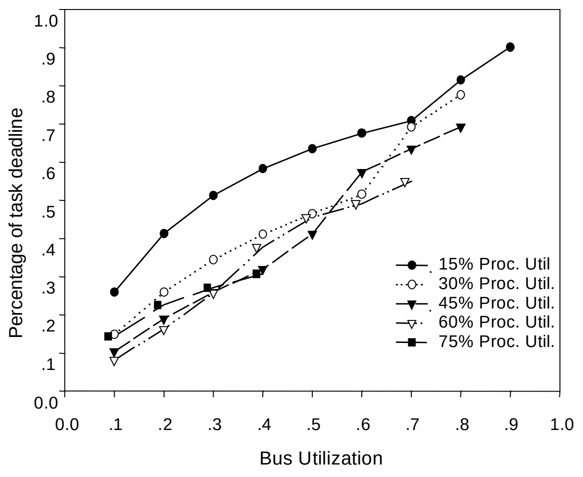


Figure 11. The ratio of message deadline to task deadline

5. Conclusion

In this paper, we present several algorithms in order to produce cyclic partition and channel schedules for the two-level hierarchical scheduling mechanism of IMA-based avionics systems. The system model of the IMA architecture supports spatial and temporal partitioning in all shared resources. Thus, applications can be easily integrated and maintained.

The main idea of our approach is to allocate a proper amount of capacity and to follow a distance constraint on partition and channel invocations. Thus, the tasks (messages) within a partition (channel) can have an inactive period longer than the blocking time of the partition (channel). Also we use a heuristic deadline decomposition technique to find feasible deadlines for both tasks and messages. To reduce bus bandwidth requirement for message transmission, we develop a heuristic channel-combining algorithm which leads to highly utilized channels by multiplexing messages of different deadlines and periods. The simulation analyses show promising results in terms of schedulability and system characteristics.

Based on the work in this paper, we have developed a scheduling tool for the IMA-based avionics systems. The tool includes additional features for practical implementations, such as time-tick based processor scheduling, non-zero context switch overhead, replication execution and transmission, incremental changes, etc. We are currently looking into different network infrastructures and communication scheduling algorithms that can be employed in the scalable IMA-based systems.

References

[1] "Design Guide for Integrated Modular Avionics," ARINC Report 651, Aeronautical Radio Inc., Annapolis, MD, Nov. 1991.

[2] "Avionics Application Software Standard Interface," ARINC Report 653, Aeronautical Radio Inc., Annapolis, MD, Jan. 1997.

[3] "Backplane Data Bus," ARINC Specification 659, Aeronautical Radio Inc., Annapolis, MD, Dec. 1993.

[4] N. Audsley, and A. Wellings, "Analyzing APEX applications," *Proc. IEEE Real-Time Systems Symposium,* Dec. 1996, pp. 39-44.

[5] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: the deadline-monotonic approach," *Eighth IEEE Workshop on Real-time Operating Systems and Software*, 1991, pp. 133-137.

[6] T. Carpenter, "Avionics Integration for CNS/ATM," *Computer,* Dec. 1998, pp. 124-126.

[7] M. Y. Chan and F. Y. L. Chin, "General schedulers for the pinwheel problem based on double-integer reduction," *IEEE Trans. on Computers*, vol. 41, June 1992, pp. 755-768.

[8] Z. Deng and J. W. S. Liu, "Scheduling real-time applications in an open environment," *Proc. IEEE Real-Time Systems Symposium*, Dec. 1997, pp. 308-319.

[9] C.-C. Han, K.-J. Lin, and C.-J. Hou, "Distance-constrained scheduling and its applications to real-time systems," *IEEE Trans. on Computers*, Vol. 45, No. 7, July, 1996, pp. 814--826.

[10] Y. H. Lee, D. Kim, M. Younis, and J. Zhou, "Partition scheduling in APEX runtime environment for embedded avionics software," *Proc. of Real-Time Computing Systems and Applications,* Oct. 1998, pp. 103-109.

[11] J. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," *Proc. IEEE Real-Time Systems Symposium,* Dec. 1992, pp. 110-123.

[12] J. Lehoczky, L. Sha, and Y.Ding, "The rate-monotonic scheduling algorithm: exact characteristics and average case behavior," *Proc. IEEE Real-Time Systems Symposium*, Dec. 1989, pp. 166-171.

[13] J. Lehoczky, "Fixed priority scheduling for periodic task sets with arbitrary deadlines," *Proc. IEEE Real-time Systems Symposium*, Dec. 1990, pp. 201-209.

[14] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *JACM*, vol. 20, No. 1, 1973, pp.46-61.

[15] K. W. Tindell, A. Burns, and A. J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-Time Systems 6(2)*, 1994, pp. 133-151.