Appeared in the Proceedings of 16th International Conference on Distributed Computing Systems, May, 1996.

Synchronization Protocols in Distributed Real-Time Systems

Jun Sun Jane Liu
Department of Computer Science,
University of Illinois, Urbana-Champaign
Urbana, IL 61801
{jsun, janeliu}@cs.uiuc.edu

Abstract

In many distributed real-time systems, the workload can be modeled as a set of periodic tasks, each of which consists of a chain of subtasks executing on different processors. Synchronization protocols are used to govern the release of subtasks so that the precedence constraints among subtasks are satisfied and the schedulability of the resultant system is analyzable. Tasks have different worst-case and average end-to-end response times when different protocols are used. In this paper, we consider distributed real-time systems with independent, periodic tasks and fixed-priority scheduling algorithms. We propose three synchronization protocols and conduct simulation to compare their performance with respect to the two timing aspects.

1. Introduction

In many real-time systems, the workload can be modeled as a set of periodic tasks [1]. Each periodic task is an infinite stream of computation requests that are released at a fixed rate. We call each request *an instance* of the task. When the context is clear, we may simply say "a task" to mean an instance of that task. Each task is typically constrained by a *relative deadline*, which is the maximum allowed *response time* of each task instance. A real-time system is said to be schedulable if the worst-case response time of every task in the system is no longer than its relative deadline.

It is well known that fixed priority scheduling is an effective means to schedule periodic tasks on a single processor and to ensure that the time constraints of the tasks are satisfied [1, 2, 3, 4]. In this approach, each task is assigned a fixed priority. At any time, the scheduler simply chooses to execute the task with the highest priority among all the tasks whose instances are released but not yet completed. A great deal of work has been done on how to assign the priorities [1, 5, 6] and how to bound the response times of tasks [1, 7, 2] for single processor systems.

In a distributed real-time system, each instance of a task may need to execute on different processors in a sequential order. An example is a *monitor* task that collects remote sensor data and displays the data on the local screen. We will refer to this example as Example 1 in later discussion. Three steps are involved in the monitor task: sampling the sensor reading on the field processor, transmitting the sample data over the communication link[1] and displaying the data on the local screen. Here we call each step a *subtask* of the monitor task, and the task is the *parent task* of the subtasks. Moreover, subtasks that have the same parent task are *sibling subtasks* of each other. In this example, the monitor task is a chain of three subtasks, *sample*, *transfer* and *display*, executing sequentially on three different processors.

We say that a task in a distributed system is periodic if its first subtask is released periodically. Whether or not the later subtasks on the chain are released periodically depends on the *synchronization protocol* used in the system. A synchronization protocol governs how subtasks are released so that their precedence constraints are satisfied and the schedulability of the resultant system is analyzable. The relative deadline of a task that consists of a chain of subtasks is the maximum allowed length of time from the release of an instance of its first subtask till the completion of the corresponding instance of its last subtask. The relative deadline of the task is often called the *end-to-end relative deadline*, or simply the *end-to-end deadline*, and its response time is called the *end-to-end response time*. We will abbreviate the latter as the *EER time* in our discussion.

We confine our attention to systems that contain independent, preemptable periodic tasks and are scheduled on a fixed priority basis. In such a system, each subtask has a fixed priority, which may or may not be the same as the priorities of its sibling subtasks. Previous studies on the priority assignment problem in distributed real-time systems can be found in [8, 9, 10]. In this paper, we are not con-

[1] In many cases, the communication links can be modeled as processors, and consequently message transmissions can be modeled as communication tasks on the "link" processors.

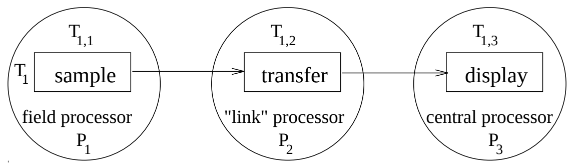

Figure 1. Example 1 - The Monitor Task with Its Three Subtasks

cerned with this problem; rather, we assume that the priorities of subtasks on each processor are given, and we focus on alternative ways to synchronize subtasks on different processors.

In this paper, we describe three synchronization protocols. The first protocol is a straightforward implementation to enforce the precedence constraints among subtasks. The second protocol is an extension to the one proposed by Bettati, which was designed for flow-shop systems with certain limitations [11]. We propose the third protocol that combines the strength of the previous two protocols while avoiding their shortcomings. We discuss the implications of these protocols with respect to two timing aspects, the estimated worst-case and average EER times of tasks, and compare the performance of these protocols based on these two timing aspects.

The rest of the paper is organized as follows. Section 2 formally presents the synchronization problem addressed in this paper, as well as the criteria used to measure the performance of synchronization protocols. Section 3 describes three synchronization protocols in detail. Section 4 compares their performance, and Section 5 concludes the paper.

2. The Problem Formulation

We consider here a distributed real-time system that consists of a set $\{P_i\}$ of processors and a set $\{T_i\}$ of independent, preemptable tasks. Each task T_i consists of a chain of n_i subtasks, $T_{i,1}, T_{i,2}, \ldots, T_{i,n_i}$, and subtasks execute on different processors. The execution time of each subtask $T_{i,j}$ is $\tau_{i,j}$. A fixed-priority scheduling algorithm is used to schedule subtasks on each processor.

Each task T_i is a periodic task, i.e., instances of $T_{i,1}$ are released periodically. Period p_i is the minimum inter-release time of instances of $T_{i,1}$. The phase of T_i is the release time of the first instance of its first subtask $T_{i,1}$. Depending on the particular synchronization protocol used in the system, instances of a later subtask $T_{i,j}$ ($j > 1$) may or may not be released periodically. For the sake of discussion, we define the period of a subtask to be equal to the period of its parent task, even if it is not released periodically.

In this paper, we do not explicitly model inter-processor communication. In some cases, such as in CAN [12], where message transmissions are prioritized, communication links can be modeled as processors, and message transmissions can be modeled as communication subtasks on "link" processors. In some other cases, such as when dedicated communication links are used, message transmissions can be either modeled as critical sections or taken into account as blocking times of the sending subtasks. For this reason, this model remains applicable even when the communication overhead is not neglectable. In this paper, we assume that the cost of inter-processor communication required to synchronize subtasks on different processors is zero. As an example, suppose that the communication link in Example 1 can be modeled as a "link" processor. The monitor task, T_1, is then modeled as a parent task of three subtasks executing on three different processors, as shown in Figure 1.

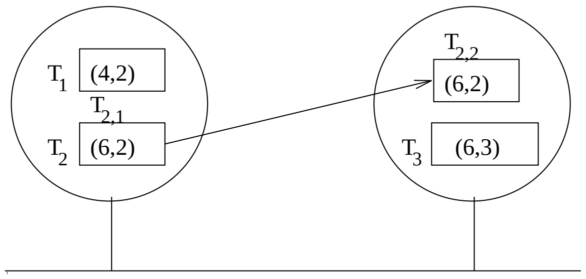

Figure 2. Example 2 - Illustration of the Synchronization Protocol Problem

To illustrate the problem dealt with by a synchronization protocol, we consider a second example, Example 2, shown in Figure 2, where the system consists of two processors, P_1 and P_2, and three periodic tasks, T_1, T_2 and T_3. Task T_1 and T_3 have only one subtask. (We use T_1 and T_3 to refer to the subtasks also.) T_2 has two subtasks, $T_{2,1}$ and $T_{2,2}$. The period and the execution time of each subtask are given in the rectangular box representing the subtask. The phases of T_1 and T_2 are zero, and the phase of T_3 is 4. On processor P_1, T_1 has a higher priority than $T_{2,1}$, and on processor P_2, $T_{2,2}$ has a higher priority than T_3. The relative deadline of each task is equal to its period.

Figure 3 shows the schedules on the two processors for the first 10 time units. In this schedule, each instance of $T_{2,2}$ is released on P_2 as soon as the corresponding instance of $T_{2,1}$ completes on P_1. This is achieved by an synchronization signal sent from P_1 to P_2 whenever an instance of $T_{2,1}$ completes, represented by a dotted arrow in the figure. As a matter of fact, this scheme is the first synchronization protocol described in the next section. We notice here that, although subtask $T_{2,1}$ is released periodically, $T_{2,2}$ is not. It is easy to verify that the instances of $T_{2,2}$ are released at times 4, 8, 16, 20, 28, $\ldots$. The first instance of T_3 is preempted by the first and the second instance of $T_{2,2}$. As a result, this instance of T_3 misses its deadline at time 10. On the other hand, if $T_{2,2}$ were released periodically every 6 time units, any instance of T_3 would be preempted at most by one instance of $T_{2,2}$, because they have the same period. Task T_3 would have a worst-case response time of 5 time units and would never miss a deadline. As we shall see later, other synchronization protocols achieve this result by controlling the releases of $T_{2,2}$. Different protocols have the different impact on the worst-case EER times of tasks, which in turn translates to different upper bounds on the EER times of tasks.

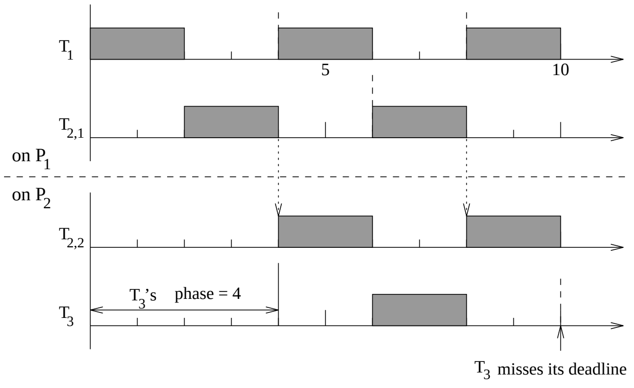

Figure 3. The Schedule of Example 2 Using the DS Protocol

In next section, we will describe three synchronization protocols. Their performance will be measured according to the following three criteria.

Implementation complexity and run-time overhead : We look for protocols that are easy to implement and have low run-time overheads.

Average EER times of tasks : For many applications, especially interactive applications, it is important for the average EER times of tasks to be as short as possible.

Estimated worst-case EER times of tasks : For each protocol, we apply the best known timing analysis algorithm to obtain an upper bound on the EER time of each task and use the upper bound as an estimate of the worse-case EER time of the task.

The estimated worst-case EER time of a task may be larger than the actual worst-case EER time since existing timing analysis algorithms do not yield tight bounds on the EER times in general. The actual worst-case EER times of tasks can be found only via exhaustive search, which is too time consuming to be practical even for small systems. Consequently, it is a common practice to determine the schedulability of a real-time system based on upper bounds of response times computed from the best known timing analysis algorithms. For this reason, we compare the protocols proposed here according to the tightest upper bounds of EER times found for the protocols. In subsequent discussion we will use the terms, the estimated worst-case EER time or the upper bound of the EER time of a task, interchangeably.

Sometimes applications also require small *output jitters*, where an output jitter is the difference in the EER times of two consecutive task instances. We will address this issue when we need to.

3. The Synchronization Protocols

We call the synchronization protocol that leads to the schedule in Figure 3 the Direct Synchronization protocol, abbreviated as the DS protocol. Again, according to this protocol, when an instance of a subtask completes, a synchronization signal is sent to the processor where its immediate successor executes, and an instance of its successor is released immediately. The DS protocol is obviously easy to implement and has the minimum necessary run-time overhead. It is also expected to yield relatively short average EER times due to its aggressiveness in releasing subtasks. However, the performance of this protocol is poor when measured in terms of the estimated worst-case EER times. Upper bounds of the EER times of tasks can be computed by an iterative algorithm described in [13], which is the only known algorithm that provides reasonably tight bounds on the EER times of tasks. As we will see later, when the subtask chains are long and the processor utilizations are high, the algorithm may fail to obtain finite bounds. In cases when we do obtain finite bounds, they are considerably larger than those yielded by other protocols.

The Phase Modification protocol and the Release Guard protocol are two alternative ways to control the releases of successor subtasks. By ensuring that instances of each subtask are not released too soon, they attempt to improve the schedulability of the tasks at the expense of their average EER times.

3.1. The Phase Modification (PM) Protocol

The Phase Modification protocol, abbreviated as the PM protocol, was initially proposed by Bettati and used to schedule periodic flow-shop tasks [11]. Unlike the DS protocol, the PM protocol insists that instances of all subtasks are released periodically according to the periods of their parent tasks. To ensure the precedence constraints among subtask are satisfied, each subtask is given its own phase. The phases of subtasks are properly adjusted as follows. Let $R_{i,j}$ denote an upper bound on the response time of subtask $T_{i,j}$. The phase of the first subtask, $f_{i,1}$, is the same as the phase of the parent task T_i. For a later subtask $T_{i,j}$ ($j > 1$), we adjust its phase to ($f_{i,1} + \sum_{k=1}^{j-1} R_{i,k}$), namely, the phase of its parent task plus the sum of the bounds on the response times of its predecessors. It can be easily shown that, if the clocks in the system are synchronized, whenever an instance of a subtask is released, the corresponding instance of its predecessor must have completed. The schedule shown in Figure 4 illustrates the application of the PM protocol to task T_1 in Figure 1. The subtasks are synchronized according to the PM protocol. We let the phase of task T_1 be 0. Hence, the phase of $T_{1,1}$ is also 0. The phases of $T_{1,2}$ and $T_{1,3}$ are set to $R_{1,1}$ and ($R_{1,1} + R_{1,2}$) respectively. All instances of the three subtasks are released periodically according to the period p_1 and their own phases.

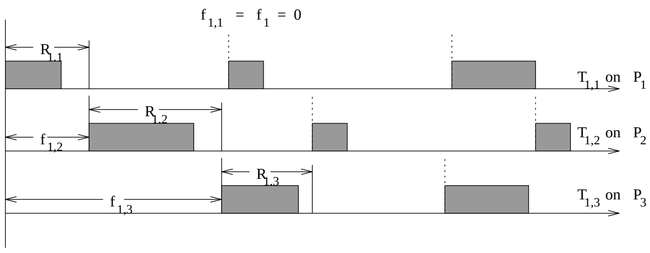

Figure 4. The Schedule of T_1 in Example 1 When the PM Protocol Is Used

Since all subtasks on each processor are strictly periodic, the Busy Period Analysis method proposed by Lehoczky [2] can be applied to obtain a bound on the response time of each subtask. The estimated worst-case EER time of a task is simply the sum of the bounds on the response times of all its subtasks. Figure 5 shows the schedule of Example 2 when the PM protocol is used. The bound on the response time of $T_{2,1}$ is 4 time units, and therefore the phase of $T_{2,2}$ is 4. In this case the first instance of T_3 meets its deadline because the second instance of $T_{2,2}$ is not released until time 10 and hence does not preempt the first instance of T_3.

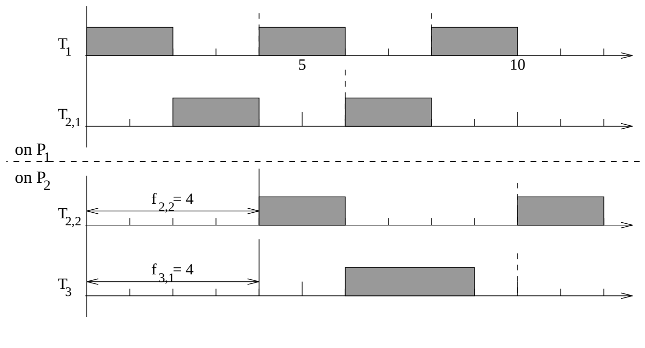

Figure 5. The Schedule of the System in Example 2 When the PM Protocol Is Used

The run-time overhead of the PM protocol involves the periodic timer interrupts that trigger the releases of subtasks. However, to guarantee that the precedence constraints among subtasks are satisfied, the PM protocol requires a centralized clock or strict clock synchronization, both of which can be difficult to achieve in practice. In addition, if a subtask occasionally overruns or the inter-release time of the first subtask of a task is greater than the period, the precedence constraints can still be violated. A slight modification of the protocol, called the Modified Phase Modification protocol or the MPM protocol, overcomes these shortcomings at a slightly higher run-time expense. To illustrate the MPM protocol, let us consider subtask $T_{i,j}$ which executes on processor P_k and whose response time is never greater than $R_{i,j}$. Suppose an instance of $T_{i,j}$ is released at time t with respect to the clock on P_k and it completes at time C. The scheduler first checks whether time C is later than or equal to $(t + R_{i,j})$. If it is, the scheduler sends a synchronization signal to the processor where subtask $T_{i,j+1}$ executes. Otherwise, the scheduler sends the synchronization signal at time $(t + R_{i,j})$. Upon receipt of such a signal, an instance of $T_{i,j+1}$ is released immediately. It is easy to verify that under the ideal conditions, i.e., clocks are synchronized, subtasks do not overrun, and the release of first subtasks are strictly periodic, the PM protocol and the MPM protocol produce identical schedules. However, the MPM protocol achieves this without requiring clock synchronization and strictly periodic release of first subtasks. It also preserves the precedence constraints among subtasks even when some subtasks overrun. Figure 6 illustrates the application of the MPM protocol to task T_1 in Figure 1. In the figure, there are several places where the response time of a subtask instance is shorter than the upper bound, and the synchronization signal is delayed. The resultant schedule is the same as the one in Figure 4.

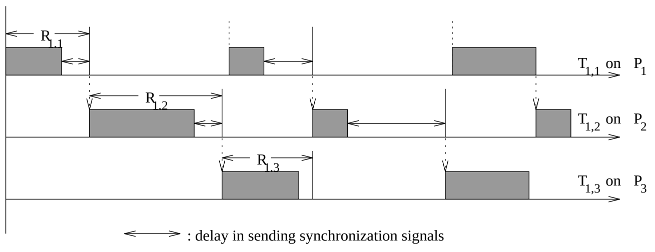

Figure 6. The Schedule of Task T_1 in Example 1 When the MPM Protocol Is Used

According to the PM protocol and the MPM protocol, each instance of subtask $T_{i,j}$ is released $R_{i,j-1}$ time units after the corresponding instance of subtask $T_{i,j-1}$ is released, no matter how soon $T_{i,j-1}$ might actually complete. While an upper bound on the EER times of a task T_i is $(\sum_{k=1}^{n_i} R_{i,k})$, the EER time of T_i is never less than

$(\sum_{k=1}^{n_i-1} R_{i,k} + \tau_{i,n_i})$. This lower bound is large because $R_{i,j}$ is normally much larger than the average response time of $T_{i,j}$. This motivates us to look for other protocols that may yield estimated worst-case EER times as small as the PM and MPM protocols and yet average EER times as short as the DS protocol.

3.2. The Release Guard (RG) Protocol

The idea behind the RG protocol is to control the releases of a subtask such that the inter-release time of any two consecutive instances is no shorter than the period. In implementation, we associate each subtask $(T_{i,j})$ with a variable $g_{i,j}$, called its *release guard,* which is the earliest allowed release time for next instance of $T_{i,j}$. If an instance of the immediate predecessor of $T_{i,j}$ completes after $g_{i,j}$, an instance of $T_{i,j}$ can be released right away. Otherwise, the instance will not be released until time $g_{i,j}$. Initially, each release guard $(g_{i,j})$ is set to 0, so that the first instance of each subtask can be released as soon as the first instance of its immediate predecessor completes. Afterwards release guards are updated according to the following two rules.

1. When an instance of subtask $T_{i,j}$ is released, update $g_{i,j}$ to the current time plus the period of $T_{i,j}$ (p_i).

2. Update $g_{i,j}$ to the current time if it is an idle point.[2]

Figure 7 shows the schedule of Example 2 if we use the RG protocol. The schedule is similar to the schedule produced by the DS protocol up until when the second instance of $T_{2,1}$ completes at time 8. We notice that the second instance of $T_{2,2}$ is not released because the release guard $(g_{2,2})$ of $T_{2,2}$ is equal to 10, which is set at time 4 when the first instance of $T_{2,2}$ is released. Thus we would expect that the second instance of $T_{2,2}$ will not be released till time 10. However, time 9 is an idle point on processor P_2, and $g_{2,2}$ gets updated to the current time, 9. Consequently, the second instance of $T_{2,2}$ is released at time 9. As we shall see later, such an early release does no harm to the worst-case response times of other subtasks, but helps reduce the average EER times of tasks.

One nice attribute of the RG protocol is that we can apply the busy period analysis method for periodic tasks [2] to obtain upper bounds on the response times of subtasks, as we do for the PM and MPM protocols. By rule (1) for updating the release guards, we see that for each subtask the inter-release time between any two consecutive instances is no shorter than the period, which makes the subtask a periodical subtask. Although the inter-release time can occasionally be shorter than the period because of rule (2), it never

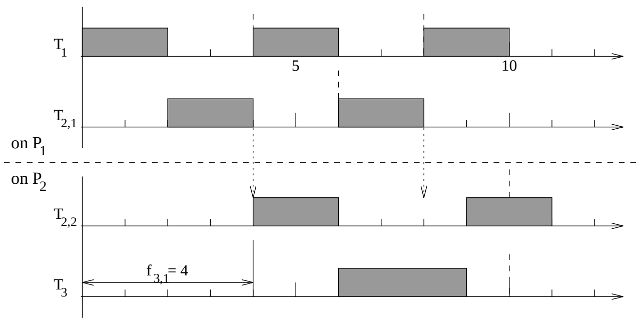

Figure 7. The Schedule of the Second Example in Figure 3 When the RG Protocol Is Used

happens during a busy period. For example, if we want to bound the response time of $T_{i,j}$. During the interval from an instance of $T_{i,j}$ is released till it completes, the processor can never be idle,[3] and hence rule (2) can never be applicable. All the interfering subtasks to $T_{i,j}$ will "behave" like periodic subtasks during that interval. Consequently, the busy period analysis for periodic tasks can be applied to obtain a upper bound on the response time of $T_{i,j}$.

It can be further shown, by induction, that the EER time of a task is also upper-bounded by the sum of the bounds on the response times of all its subtasks. Consequently, we obtain the same upper bounds on the EER time of each task when the RG protocol is used as when the PM or MPM protocol is used. Meanwhile we expect that tasks have shorter average EER times when the RG protocol is used, because the processor is never left idle if there are any pending subtask instances to be released. In addition, the subtasks are released earlier because the release times depend on the actual response times of their predecessors instead of upper bounds on the response times of their predecessors.

4. Comparison of Performance

To compare the performance of the DS, PM(MPM) and RG protocols we generated synthetically a set of representative distributed, real-time workloads. For each of them, we first use the best known algorithm to obtain estimated worst-case EER times of tasks for each of the synchronization protocols. Then, we simulate the actual execution of these systems when each protocol is used and obtain estimates of average EER times of tasks. The protocols are compared based on these two criteria.

[2] An idle point is a time instant by which all subtasks that are released before the instant have completed. Intuitively, it is a time instant when the processor is idle, unless some subtasks are just released at the same instant.

[3] Obviously we assume that subtasks do not suspend themselves.

4.1. Generation of Workloads

Through preliminary experiments, we identified two system parameters that influence the performance of synchronization protocols most: the number of subtasks in each task and the utilization of each processor. To simplify the timing analysis, we let every task in the system have the same number of subtasks and every processor have the same utilization. Each *configuration* is a unique combination of the number of subtasks in each task and the utilization of each processor, denoted by a 2-tuple (N, U). For example, configuration $(5, 60)$ represents systems where each task has 5 subtasks and the utilization of each processor is 60%. In the configurations evaluated, the number of subtasks in each task ranges from 2 to 8 and the utilization of each processor are 50%, 60%, 70%, 80%, or 90%. Consequently, we have a total of 35 configurations.

For each of the configuration, we generated 1000 systems. Each system has 4 processors and 12 tasks. (Again, we found that the performance of the protocols are not sensitive to these parameters. These values were chosen to keep the simulation time from becoming impractically large.) The periods of tasks are exponentially distributed from 100 to 10000. This yields task periods with more variation than when the periods are evenly distributed in $(100, 10000)$. Subtasks are randomly assigned to processors, with no two consecutive subtasks of the same parent task assigned to the same processor. Subtasks on each processor divide the utilization of the processor randomly. Specifically, to determine the utilization of a subtask, we generate a random number from 0.001 to 1 for each subtask. The utilization of the subtask is equal to the total processor utilization times the ratio of the random number and the sum of the random numbers of all the subtasks on the same processor. The execution time of the subtask is equal to its utilization times its period.

The last parameter of each system is priority assignment of subtasks. We choose the Proportional-Deadline-Monotonic priority assignment method to assign priorities to subtasks. (A similar method is called the Equal Flexibility assignment in [9].) According to this method, each subtask has a proportional deadline ($PD_{i,j}$) defined as follows.

$$PD_{i,j} = \frac{\tau_{i,j}}{\sum_{k=1}^{n_i} \tau_{i,k}} D_i$$

where D_i is the relative deadline of T_i, which is equal to its period for this simulation. A subtask has higher priority if it has a shorter proportional deadline.

4.2. Comparison of the Estimated Worst-Case EER Times of Tasks

The PM, MPM and RG protocols have the same bounds on the EER times of tasks. Hence in the subsection we will only compare the PM protocol and the DS protocol.

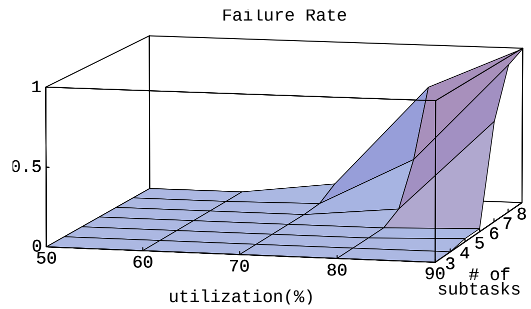

Figure 8. The Failure Rates as a Function of Configurations for the DS Protocol

Again, when tasks are synchronized according to the PM protocol, we can compute the estimated worst-case response times of the tasks using a straight forward extension of the Busy Period Analysis method [2]. We have developed an iterative algorithm to bound the EER times of tasks synchronized according to the DS protocol. Due to space limitations, the algorithm is not described here; it can be found in [13]. In general, the estimated worst-case EER times of tasks are larger when tasks are synchronized according to the DS protocol than when tasks are synchronized according to the PM protocol. For some systems, the bounds on the EER times of tasks under the DS protocol were found to be extremely large. We say that a failure occurs when we find that the upper bound of the EER time of a task is larger than 300 times of its period and hence for all practical purposes equals to infinity. (We say that the bound is infinite in this case.) By analyzing the systems generated in the way described above, we determined under what conditions (1) the estimated worst-case EER times for these two protocols are close, (2) the estimated worst-case EER times of tasks for the DS protocol are much greater, and (3) failures occur when the DS protocol is used.

Figure 8 shows the *failure rate*, which is the percentage of systems that we fail to obtain finite bounds on the EER times when the DS protocol is used as a function of the system configuration. We notice that the failure rates are mostly zero and quickly rise to 1 when number of subtasks in each task approaches 8 and the processor utilizations are close to 90%. In the case of configuration $(8, 90)$, we could obtain finite bounds for only 4 out of 1000 systems. Similarly, we observe that failure rates are greater than 0.1 for configuration $(8, 80)$, $(7, 90)$, $(7, 80)$, and $(6, 90)$. This indicates that the DS protocol cannot be applied for this kind of systems.

For the systems that have finite estimated worst-case EER times for both the PM protocol and the DS protocol, we use the average *bound ratio* to compare their perform-

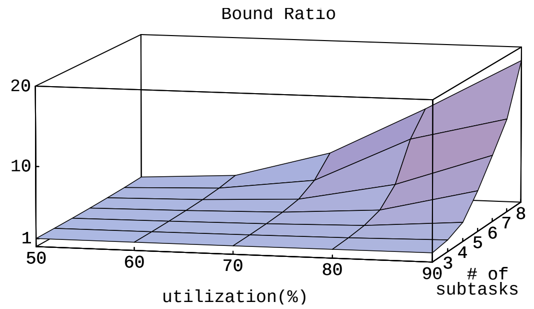

Figure 9. Bound Ratios as a Function of Configurations

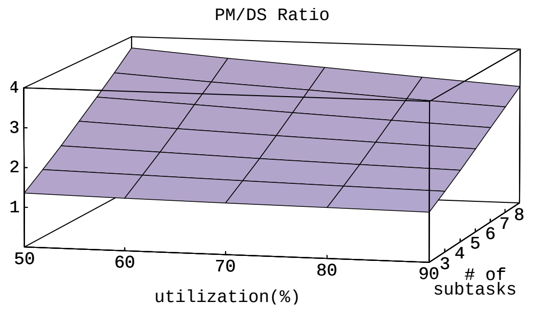

Figure 10. The PM/DS Ratio

ance. A *bound ratio* is the ratio of the estimated worst-case EER time of a task synchronized according to the DS protocol to the estimated worst-case EER time of the task under the PM protocol. For each configuration, we average the bound ratios of all the tasks in the systems that have finite estimated worst-case EER times for the DS protocol. A large average bound ratio for a configuration indicates that the DS protocol performs poorly. The average bound ratios as a function of configurations is plotted in Figure 9. The 90% confidence interval is negligibly small for most configurations. Although they seem to fit the overall curves well, the average bound ratios for configurations with large number of subtasks and high utilizations are not statistically significant due to the high failure rates they have.

From Figure 9, we see that for low utilization configurations, the ratios stay relatively equal as the number of subtasks in each task increase. However, for high utilization configurations, the ratios go up quickly as the number of subtasks in each task increases. Roughly one-third of configurations have the ratios greater than 2, making the DS protocol less appealing for these kinds of systems.

4.3. Comparison of the Average EER Times of Tasks

We simulated the execution of each system to obtain the average EER times of tasks. For each system, we randomly assign phases to tasks. We then compare the relative performance between two protocols according to the ratio of the average EER times of the same task yielded by two different protocols. A *PM/DS ratio* is the average EER time of a task yielded by the PM protocol divided by the average EER time of the task yielded by the DS protocol. Similarly, the *RG/DS ratio* gives the performance comparison between the RG protocol and the DS protocol. The PM/DS, RG/DS ratios as functions of configurations are plotted in Figures 10 and 11. The 90% confidence intervals are negligibly small for all configurations.

We notice that for a fixed number of subtasks in each task, the PM/DS ratios goes down when the utilization goes up on each processor. This is because with low utilizations many idle intervals in the schedules are caused by the PM protocol when the releases of subtasks are postponed according to the protocol. Therefore the average EER times are larger than those yielded by the DS protocol. The PM/DS ratio increases as the number of subtasks in each task increases. For configurations with 5 or more subtasks in each task, the values of the ratio are all greater than 2, indicating that for these kinds of configurations, the average EER times of tasks synchronized according to the PM protocol are more than twice than when tasks are synchronized according to the DS protocol. The slope of the increase, however, is rather flat. For configurations with 8 subtasks in each task, the ratio is around 3 or 4.

The RG/DS ratio varies mostly from 1 to 2 for all configurations, except for some configurations with 90% utilization on each processor. When the processors are busy almost all the time, the releases of subtasks according to the RG protocol become more periodical. The intentional delays in releases of subtasks in this case lead to longer average EER times. Overall, the RG protocol performs much better than the PM protocol with respect to the average EER times of tasks.

5. Conclusions

In this paper, we proposed three synchronization protocols and evaluated their performance for different system configurations. Among the protocols, the DS protocol has low complexity and overhead and yields short average EER times. It is a reasonable choice for applications with soft timing constraints, tasks with short subtask chains, or low processor utilizations. However, the results of our timing analysis and simulation indicate that for applications that

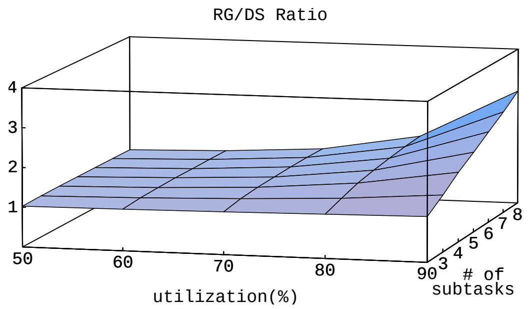

Figure 11. The RG/DS Ratio

have high processor utilization, a long subtask chain in each task, and hard timing constraints, the DS protocol is not a suitable choice because it leads to large, or even unbounded, worst-case EER times. The PM protocol and the RG protocol are better for these applications. Specifically, the RG protocol is superior to the PM and MPM protocols because it yields reasonably short average EER times of tasks. However, the output jitters yielded by the RG protocol can be as large as the estimated worst-case EER time of a task, which is much greater than the maximum output jitter yielded by the PM or MPM protocol, which is equal to the estimated worst-case response time of the last subtask in a task. The PM or MPM protocol should be favored when small output jitters are desirable.

In previous studies on scheduling distributed real-time applications, such as in [14], subtasks are typically assigned local deadlines and scheduled locally. Loose synchronization among subtasks is assumed. In this paper, we have attempted to provide insight to the synchronization problem in distributed real-time systems and an initial answer towards an unified end-to-end scheduling framework. Much work remains to be done. For example, the synchronization protocols described here, as well as the timing analysis algorithms for them, assume that the variations in the execution times of subtasks and jitters in the task release times are small. Algorithms that can effectively deal with wide variations in these parameters are needed.

References

[1] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[2] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

[3] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[4] K. W. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems*. PhD thesis, University of York, Department of Computer Science, 1994.

[5] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[6] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Dept. of Computer Science, University of York, December 1991.

[7] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal of the British Computer Society*, 29(5):390–395, October 1986.

[8] K. Tindell, A. Burns, and A. Wellings. Allocating real-time tasks. An NP-hard problem made easy. *Real-Time Systems Journal*, 4(2), May 1992.

[9] B. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. In *The 13th International Conference on Distributed Computing Systems*, pages 428–437, May 1993.

[10] J. Garcia and M. Harbour. Optimized priority assignment for tasks and messages in distributed hard real-time systems. In *The Third Workshop on Parallel and Distributed Real-Time Systems*, pages 124–132, April 1995.

[11] R. Bettati. *End-to-End Scheduling to Meet Deadlines in Distributed Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 1994.

[12] B. Hunting. The solution's in the CAN — Part 1. *Circuit Cellar*, pages 14–20, May 1995.

[13] J. Sun and J. Liu. Bounding the end-to-end response times of tasks in a distributed real-time system using the direct synchronization protocol. Technical Report UIUCDCS-R-96-1949, University of Illinois at Urbana-Champaign, Dept. of Computer Science, March 1996.

[14] S. Chatterjee and J. Strosnider. Distributed pipeline scheduling: End-to-end analysis of heterogeneous, multi-resource real-time systems. In *The 15th International Conference on Distributed Computing Systems*, pages 204–211, May 1995.