Period Optimization for Hard Real-time Distributed Automotive Systems

Abhijit Davare[1], Qi Zhu[1], Marco Di Natale[2],
Claudio Pinello[3], Sri Kanajan[2], Alberto Sangiovanni-Vincentelli[1]

[1] University of California, Berkeley
[2] General Motors Research
[3] Cadence Berkeley Labs

ABSTRACT

The complexity and physical distribution of modern active-safety automotive applications requires the use of distributed architectures. These architectures consist of multiple electronic control units (ECUs) connected with standardized buses. The most common configuration features periodic activation of tasks and messages coupled with run-time priority-based scheduling. The correct deployment of applications on such architectures requires end-to-end latency deadlines to be met. This is challenging since deadlines must be enforced across a set of ECUs and buses, each of which supports multiple functionality. The need for accommodating legacy tasks and messages further complicates the scenario.

In this work, we *automatically* assign task and message periods for distributed automotive systems. This is accomplished by leveraging schedulability analysis within a convex optimization framework to simultaneously assign periods and satisfy end-to-end latency constraints. Our approach is applied to an industrial case study as well as an example taken from the literature and is shown to be both effective and efficient.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-based Systems**]: Real-time and embedded systems

General Terms

Algorithms, Performance

Keywords

automotive systems, activation period, end-to-end latency

1. INTRODUCTION

In the automotive domain, modern active-safety applications consist of complex end-to-end computations that collect data from 360^o sensors around the vehicle to understand the positioning of surrounding objects and detect hazardous conditions. On hazard detection, active safety functions attempt to inform the driver or provide control overlays to reduce the risk. Most of these functions are high-level controls which drive low-level actuation loops, but they are nevertheless subject to timing constraints.

Such active-safety applications are typically run on distributed architectures. Distributed architectures supporting the execution of hard real-time applications are common not only for automotive, but also for avionics and industrial control systems. To provide design-time guarantees on timing constraints, different design and scheduling methodologies are used. For instance, avionics systems are often built based on static, time-driven schedules. Due to resource efficiency and ultimately price concerns, many automotive systems are designed based on *run-time priority-based scheduling* of tasks and messages. Examples of standards supporting this model are the OSEK operating system standard [13] and the CAN bus arbitration model [2].

Different interaction models may be implemented at the interface between any two resource domains (such as an ECU and a bus). The simplest interaction model consists of the *periodic activation* with *asynchronous communication*, where all interacting tasks are activated periodically and communicate by means of asynchronous buffers based on non-blocking read/write semantics. Similarly, message transmission is triggered periodically and each message contains the latest values of the signals that are mapped into it.

More specifically, the execution model considered in this work is the following. Input data (generated by a sensor, for instance) is available at one of the system's ECUs. A periodic activation signal from a local clock triggers the computation of an application task on this ECU. Local clocks on different ECUs are not synchronized. The task reads the input data, computes intermediate results, and writes them to the output buffer from where they can be read by another task or used for assembling the data content of a message. Messages - also periodically activated - transfer the data from the output buffer on the current ECU over the bus to an input buffer on another ECU. Tasks may have multiple fan-ins and messages can be multicast. Eventually, task outputs are sent to a system output (an actuator, for instance).

When implementing feedback control applications in this fashion, the (quasi) periodic stream of actuator commands may be based on sensor data taken a variable number of samples in the past, depending on how the various clocks align. For this reason, the control algorithms are typically designed favoring robustness over performance. Techniques like time-stamping and sequence counters are sometimes used at the application level to compensate for variations and to improve robustness. Nonetheless, hard bounds on latency and periodicity are provided as implementation requirements.

DAC 2007, June 4–8, 2007, San Diego, California, USA.

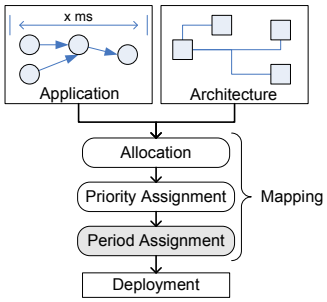

Figure 1: Period assignment within the overall design flow

1.1 Design Flow

The period assignment problem addressed in this work tackles a part of the larger design flow shown in Figure 1. The design flow is based on the Y-chart approach [8], where the application description and architectural description are initially separated, and joined together later in an explicit mapping step. In the application, nodes represent function blocks and edges represent data dependencies, which consist of signal information. The application description is further characterized by end-to-end latency constraints along certain paths from sources to sinks. The architectural description is a topology consisting of ECUs connected with buses. In this work, the ECUs are assumed to run OSEK-compliant operating systems which have preemptive priority-based run-time task scheduling. The buses use the standard CAN bus arbitration model, which features non-preemptive priority-based run-time message scheduling.

Mapping deploys functional blocks to tasks and tasks to ECUs. Correspondingly, signals are mapped into local communication or messages that are exchanged over the buses. We further divide the mapping step into three stages: allocation, priority assignment, and period assignment. Allocation is the first stage and assigns tasks to ECUs and messages to buses. Each task is allocated to a single ECU while each message is allocated to a single bus. The second stage assigns priorities to both tasks and messages. The last stage assigns periods to task and messages.

In this work, we restrict the focus to the period assignment stage. Given an allocation and priority assignment for both tasks and messages, our approach automatically assigns periods for all tasks and messages in order to satisfy the end-to-end latency requirements. The results of period assignment may trigger design iterations over the allocation and/or priority assignment stages when a feasible solution cannot be found or when the design can be further improved by changing the allocation or reassigning the priorities (for example, following the rate monotonic rule).

1.2 Prior Work

Both static and dynamic priority, distributed as well as centralized scheduling methods have been proposed in the past for distributed systems. Static and centralized scheduling is typical of time triggered design methodologies, like the Time-Triggered Architecture (TTA) [10] and its network protocol TTP and of implementations of synchronous reactive models, including Esterel and Lustre [1]. Also, the recent FlexRay standard [5] for high speed communication in automotive systems provides two transmission windows, one dedicated to time-driven periodic streams with static design-time assignment of transmission slots, and the other for asynchronous event-driven communication.

Priority-based scheduling is also very popular in control applications and is supported by the native CAN network arbitration protocol. The worst case transmission latencies of CAN messages (with timing constraints) have been analyzed and discussed in past research work [18]. Also, the OSEK operating system standard for automotive applications supports not only priority scheduling, but also resource sharing with predictable blocking times. Priority-based scheduling of single processor systems has been thoroughly analyzed with respect to worst case response time and feasibility conditions [7].

End-to-end deadlines have been discussed in research work in the context of both single-processor as well as distributed architectures. The synthesis of task parameters (activation rates and offsets) and (partly) of task configuration itself in order to guarantee end-to-end deadlines in single processor applications is discussed in [6]. Later, the work has been tentatively extended to distributed systems [16] where a set of design patterns are applied to meet the deadlines using offset-based scheduling.

The periodic activation model with asynchronous communication can be analyzed quite easily in the worst case, because it allows the decomposition of the end-to-end schedulability problem into local problem instances, one for each resource (ECU or bus). This is not true in the case of data-driven activation models, where local schedulers have cross dependencies due to the propagation of activation signals. In this case, the problem of distributed hard real-time analysis has been first addressed by the holistic model [14] based on the propagation of the release jitter along the computation path.

While the prior work provides analysis procedures with reduced pessimism, the synthesis problem is today largely open, except for [15], where the authors discuss the use of genetic algorithms for optimizing allocation and priority assignments with respect to a number of constraints, including end-to-end deadlines and jitter.

2. REPRESENTATION

The systems we consider can be represented as a weighted directed graph ($\mathcal{O}$,$\mathcal{L}$) and a set $\mathcal{R}$. $\mathcal{O}$ is the set of vertices denoting the schedulable objects (tasks and messages), $\mathcal{L}$ is the set of edges representing the flow of information (data dependencies), and $\mathcal{R}$ is a set of shared resources supporting the execution of the tasks (ECUs) and the transmission of messages (buses).

- $\mathcal{O} = \{o_1, \ldots, o_n\}$ is the set of schedulable *objects* implementing the computation and communication functions of the system. An object o_i represents either a task or a message and is characterized by two parameters: a maximum time requirement c_i and a resource R_j to which it is allocated ($o_i \rightarrow R_j$). All objects are scheduled according to their priority and a total order exists between the priorities of all objects on each resource. The object is periodically activated with a period t_i. r_i is the worst case response time of o_i, representing the largest time interval from the activation of the object to its completion in case it is a task, or its arrival at the destination in case it is a message. The response time of an object includes its own time requirement as well as the time spent waiting to gain access to the resource.
- $\mathcal{L} = \{l_1, \ldots, l_m\}$ is the set of *links*. A link $l_i = (o_h, o_k)$ connects an object o_h (the source) to object o_k (the sink). One object can be the source or sink of many links. At the end of its execution or transmission, an object delivers results (task) or its data content (message) on all outgoing links. For any link, the sink object is activated by a periodic timer and, when it executes, reads the latest signal value that was transmitted over the link.

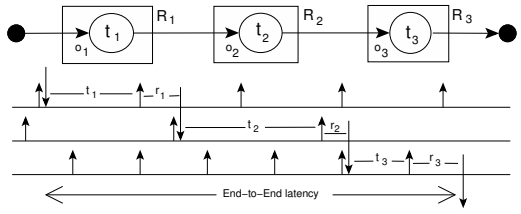

Figure 2: End-to-End Latency Calculation

- $\mathcal{R} = \{R_1, \ldots, R_z\}$ is the set of logical resources that can be used by the objects to carry out their computations. Resources are either ECUs or buses and are scheduled with a priority-based scheduler.

A *path* p is a finite sequence of objects ($p \in \mathcal{O}^*$) that, starting from $o_i = src(p)$, reaches $o_j = snk(p)$ with a link between every pair of adjacent objects. o_i is the path's source and o_j is the sink. Sources are activated by external events, while sinks activate actuators. Multiple paths may exist between each source-sink pair. The worst case end-to-end latency incurred when traversing a path p is denoted as ℓ_p. The *path deadline* for p, denoted by d_p, is an application requirement that may be imposed on selected paths.

2.1 End-to-End Latency

The worst case end-to-end latency can be computed for each path by adding the worst case response times and the periods of all the objects in the path:

$$\ell_p = \sum_{k:o_k \in p} t_k + r_k$$

In the worst case, as shown in Figure 2, an external event arrives immediately after the completion of the first instance of task o_1. The event data will be read by the task on its next instance and the result will be produced after its worst case response time, that is, $t_1 + r_1$ time units after the arrival of the external event. Since there is no coordination between tasks on separate resources, the situation repeats in the worst case for each link in the path. To get more precise results, the best case response time v_i of any predecessor object o_i should be subtracted from the period t_i in the previous formula. However, in most cases, including the case studies in Section 4, $v_i \ll t_i$ and v_i can be ignored.

For multiple communicating tasks with harmonic periods on the same ECU, the analysis can be less pessimistic if we assume that the designer can select the relative activation phase of all tasks. In case the sink task is activated with a relative phase with respect to the source equal to its worst case response time, then the contribution of the pair to the end-to-end latency can possibly be reduced. Let o_1 and o_2 be two tasks on the same ECU that appear (in that order) in a path with an end-to-end deadline. If $t_1 = kt_2$ is satisfied, where $k \in \mathbb{N}^+$, then t_2 is *oversampled-harmonic* with respect to t_1. Similarly, if $kt_1 = t_2$, where $k \geq 2$, then t_2 is *undersampled-harmonic* with respect to t_1. Latency analysis for these situations is developed in [12] and summarized in Table 1.

Condition	Path Fragment Latency
Non-local or non-harmonic	$r_1 + t_1 + r_2 + t_2$
Local oversampled-harmonic	$r_1 + t_1 + r_2$
Local undersampled-harmonic	$r_1 + r_2 + t_2$

Table 1: Latency over local harmonic path fragments

2.2 Response Time Analysis

The key in adjusting object periods to meet end-to-end latency constraints is determining the relationship between object periods and response times. The relationships are similar, but not identical, for tasks and messages. The analysis in this section summarizes work from [7, 18].

2.2.1 Task Response Times

In a system with preemption and priority-based scheduling, the worst case response time r_i for a task $o_i \in T$ depends on the computation time requirement c_i for the task itself as well as the interference from higher priority tasks on the same resource. r_i can be calculated using the following formula:

$$r_i = c_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{t_j} \right\rceil c_j \qquad \forall o_i \in \mathcal{T} \tag{1}$$

Where $j \in hp(i)$ refers to the set of higher priority tasks on the same resource. Note that the term $\lceil \frac{r_i}{t_j} \rceil$ indicates the maximum number of preemptions from a higher priority task j.

2.2.2 Message Response Times

Worst case message response times are calculated similarly to worst case task response times. The main difference is that message transmission on the CAN bus is not preemptable. Therefore, a message o_i may have to wait for blocking time b_i, which is $\max_{j \in lp(i)} c_j$ where $lp(i)$ is the set of lower priority messages that are allocated to the same bus as o_i. Likewise, the message itself is not subject to preemption from higher priority messages during its own transmission time c_i. The response time relationship is:

$$r_i = c_i + b_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i - c_i}{t_j} \right\rceil c_j \qquad \forall o_i \in \mathcal{M} \tag{2}$$

3. PERIOD OPTIMIZATION APPROACH

From the relationships given in Equations 1 and 2, it is apparent that the response time of each object is related to the periods of higher priority objects on the same resource. Intuitively, reducing the period of an object will increase the response times of other objects with lower priorities on the same resource. The end-to-end latencies of multiple paths may be affected as a result.

If object periods are modified individually, then achieving convergence is difficult, since any change to one period affects many others. Instead, we concentrate on mathematical programming (MP) techniques, which simultaneously consider modifications to the periods of all objects.

In MP, the system is represented with parameters, decision variables, and constraints over the parameters and decision variables. An objective function is defined over the same set of variables. Generic solvers can be utilized to find the optimal solution. The complexity of finding the optimal solution depends upon the variable types as well as the form of the objective function and constraints.

The benefits of a MP optimization approach are particularly relevant to the period synthesis problem. First, in assigning periods, there are a large number of interdependencies between the objects on different paths. Considering one path at a time is not guaranteed to find a feasible, let alone optimal, solution. MP approaches consider all constraints simultaneously. Next, and more importantly, MP approaches can be customized with system-specific issues by simply adding additional constraints. Whereas other solution mechanisms are brittle to changes in the problem assumptions, MP approaches can adapt to different problem assumptions

or partial solutions. For example, the existence of legacy tasks and messages whose periods are fixed or otherwise restricted can be handled quite easily with additional constraints.

The main difficulty with using MP approaches lies in finding a formulation that is sufficiently accurate to capture the behavior of the system and yet remains amenable to efficient solving.

This section is organized as follows. First, two specialized forms of mathematical programming - geometric programming (GP) and mixed-integer geometric programming (MIGP) are described in Section 3.1. The period optimization problem is defined as an MIGP in Section 3.2. A GP approximation is developed in Section 3.3 and approximation error is reduced by an iterative procedure described in Section 3.4.

3.1 Geometric Programming

Geometric programming (GP) is a special form of convex programming [4]. GPs have polynomial time computational complexity and can be solved very efficiently by a variety of off-the-shelf solvers. After [3], a GP in standard form is:

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 1 \quad i = 1, \ldots, m \\ & g_i(x) = 1 \quad i = 1, \ldots, p \end{array}$$

where $x = (x_1, ..., x_n)$ is a vector of positive real-valued decision variables. f is a set of *posynomial* functions, while g is a set of *monomial* functions. A posynomial is the sum of monomials, where a monomial function m has the following form:

$$m(x) = c x_1^{a_1} x_2^{a_2} \ldots x_n^{a_n} \qquad c > 0, a_i \in \mathbb{R}$$

If x contains both integral and real-valued decision variables, the resulting problem is a mixed-integer geometric program (MIGP). Unlike GPs, MIGPs are not convex and cannot be efficiently solved.

In this work, we make use of the gpposy [9] solver to solve GPs. Solver interfacing is handled by the Yalmip [11] framework, which can overlay a branch-and-bound approach to solve MIGP problems as well.

3.2 Problem Definition

$$min. \quad \sum_{o_i \in \mathcal{O}} r_i \tag{3}$$

$$s.t. \quad \frac{\ell_p}{d_p} \leq 1 \qquad \forall p \in \mathcal{P} \tag{4}$$

$$\frac{c_i + \sum_{j \in hp(i)} z_{ij} c_j}{r_i} \leq 1 \qquad \forall o_i \in \mathcal{T} \tag{5}$$

$$\frac{c_i + b_i + \sum_{j \in hp(i)} z_{ij} c_j}{r_i} \leq 1 \qquad \forall o_i \in \mathcal{M} \tag{6}$$

$$\frac{r_i}{t_i} \leq 1 \qquad \forall o_i \in \mathcal{O} \tag{7}$$

$$\sum_{i: o_i \to R_j} \frac{c_i}{t_i \times u_j} \leq 1 \qquad \forall R_j \in \mathcal{R} \tag{8}$$

$$\frac{n_i}{t_i} \leq 1 \qquad \frac{t_i}{x_i} \leq 1 \qquad \forall o_i \in \mathcal{O} \tag{9}$$

$$\frac{r_i}{t_j \times z_{ij}} \leq 1 \qquad \forall o_i \in \mathcal{T} \tag{10}$$

$$\frac{r_i}{t_j \times z_{ij} + c_i} \leq 1 \qquad \forall o_i \in \mathcal{M} \tag{11}$$

The period assignment problem is defined over the following sets: the objects $\mathcal{O}$, which are partitioned into messages $\mathcal{M}$ and tasks $\mathcal{T}$, the set of resources $\mathcal{R}$, and the paths with end-to-end constraints $\mathcal{P}$. All objects $o_i \in \mathcal{O}$ have associated computation time parameters c_i, lower bounds on periods n_i, and upper bounds on periods x_i. Additionally, messages $o_i \in \mathcal{M}$ have associated blocking times b_i. Path deadlines d_p are specified for all $p \in \mathcal{P}$. u_j are the maximum permitted utilization values for all resources $R_j \in \mathcal{R}$. The main decision variables for all $o_i \in \mathcal{O}$ are the periods t_i while the response times r_i and interferences $z_{ij} \in \mathbb{Z}^+$ are used as helper variables.

The objective function can be selected according to the optimization goals. (3) corresponds to the minimization of average response time over all objects in the system. However, a different choice related to the extensibility of the solution can also be used. For instance, minimizing the maximum resource utilization.

Path latencies are met by (4). Response times are related to computation times and periods by (5) and (6), following the relationships from (1) and (2) respectively.

(7) ensures that there is no queuing of jobs, i.e. response times are lower than object periods. Resource utilization is bounded by (8). Minimum and maximum execution periods of tasks and messages may be specified separately – especially for feedback control applications – with (9).

Finally, the number of interferences z_{ij} (from a higher priority object j to a lower priority object i on the same resource) for tasks and messages are specified with (10) and (11). Note that the integrality of the z_{ij} variables causes the problem to be an MIGP.

Depending on system-specific situations, additional constraints may be added that relate the periods of different objects. For instance, periods for two objects o_i and o_j may be constrained to be equal, i.e. $t_i = t_j$, or with a given oversampling ($t_i = nt_j$) or undersampling ($mt_i = t_j$) ratio (where n and m are positive integer constants). A more generic requirement might be to ensure that the objects are undersampling or oversampling with some unknown integer proportionality k between the periods. For example, $t_i = kt_j$ where $k \in \mathbb{Z}^+$. If such constraints are defined over adjacent tasks on the same resource, the less conservative analysis from Table 1 can be used.

3.3 Approximation

Since MIGP problems are very difficult to solve, we approximate the MIGP period optimization problem with a GP formulation. In order to cast the problem into a GP form, the interference variables z_{ij} are relaxed to real-valued variables and parameters $0 \leq \alpha_{ij} \leq 1$ are added to them. For clarity, let the approximated response time variables be s_i. (10) and (11) from the MIGP become:

$$\frac{s_i}{t_j(z_{ij} + \alpha_{ij})} \leq 1 \qquad \forall o_i \in \mathcal{T} \tag{12}$$

$$\frac{s_i}{t_j(z_{ij} + \alpha_{ij}) + c_i} \leq 1 \qquad \forall o_i \in \mathcal{M} \tag{13}$$

Thus, the GP approximation consists of the objective function (3) with s_i in place of r_i, constraints (4)-(9) (also with s_i in place of r_i) and constraints (12) and (13).

If the values of all α_{ij} are 1, then the approximation is always conservative, i.e. $s_i \geq r_i$. If some $\alpha_{ij} < 1$, no such guarantees can be made. Clearly, the accuracy of the approximation depends upon the α parameters that are used.

3.4 Reducing Approximation Error

The α parameters in the GP formulation represent the degree of conservatism used for the approximation of the response times. Setting all $\alpha_{ij} = 1$ is a safe, but pessimistic approximation that may produce an infeasible problem instance. In this section, an iterative procedure is presented to find α parameters that preserve feasibility with reduced conservatism.

Given some set of α parameters, if the GP is feasible, optimal t_i values from the GP solution can be obtained. We can obtain the r_i values by substituting these t_i values into (1) and (2). For all

$o_i \in \mathcal{O}$, let e_i represent the relative error between the estimated and actual response times, i.e. $e_i = \frac{s_i - r_i}{r_i}$. If all $e_i \geq 0$, then the optimal GP solution results in a feasible solution to the exact problem, while if all $e_i = 0$, then the GP solution is not only feasible, but optimal. If some $e_i < 0$, then the GP has underestimated some response times and (4) or (7) in the exact problem may have been violated.

An iterative procedure can be used to assign the α parameters. A new GP problem is solved during each iteration, and the e_i values are used to recalculate the α parameters for the subsequent iteration. The procedure is summarized in Algorithm 1.

Algorithm 1 ITERATIVE PERIOD ASSIGNMENT PROCEDURE

```
1: Input Parameter = f // acceptable error bound
2: ∀o_i ∈ O, α_ij = 1
3: while (true) do
4:    (s, t) = GP(α) // solve the GP
5:    if infeasible then
6:       ∀o_i ∈ O, α_ij = (1/2) α_ij
7:    else
8:       vior = 0, viol = 0
9:       for all o_i ∈ O do
10:         calculate r_i
11:         e_i = (s_i − r_i) / r_i
12:         if (r_i > t_i) then vior = vior + 1
13:         α_ij = α_i − e_i
14:         ensure 0 ≤ α_ij ≤ 1
15:      ∀p ∈ P, if ℓ_p > d_p then viol = viol + 1
16:      if viol = 0 ∧ vior = 0 ∧ (∀o_i ∈ O, max(|e_i|) < f) then exit
```

The input parameter to the procedure is f, which represents the maximum permissible estimation error. At initialization, all α_{ij} are conservatively assigned to 1. Inside the loop, the GP problem is solved and the estimated response times and assigned periods are obtained. If the problem is infeasible, then all α values are scaled, and a new GP problem is solved during the next iteration. If the GP problem in the current iteration is feasible, then the exact response times are calculated with (1) and (2). The relative error e_i and possible violations to (7) can then be calculated. Next, α_{ij} values are adjusted based on e_i, and are saturated either at 0 or 1 if necessary. After all exact response times have been calculated, violations to path constraints (4) can be checked. If none of the constraints have been violated, and if the maximum absolute estimation error is lower than the limit for all objects, the procedure terminates, otherwise the next iteration is executed with the modified α values. An iteration limit may also be specified.

4. CASE STUDIES

The period optimization approach is validated in this section with two case studies. The first is an experimental vehicle system that incorporates advanced active safety features while the second is a fault tolerant distributed system taken from [17].

4.1 Active Safety Vehicle

The architecture consists of 29 ECUs connected with 4 CAN buses, with speeds ranging from 25kb/s to 500kb/s. The vehicle supports advanced distributed functions with end-to-end computations collecting data from 360° sensors to the actuators, consisting of the throttle, brake and steering subsystems and of advanced HMI (Human-Machine Interface) devices. A total of 92 tasks are executed on the ECU nodes, and 196 messages are exchanged over the four buses. Worst case execution time estimates have been obtained for all tasks. Message length and bus speed is used to calculate the maximum transmission time for all CAN messages. Each ECU is allocated from 1 to 22 tasks and each CAN bus is allocated from 14 to 105 messages. The system graph contains a total of 604 links.

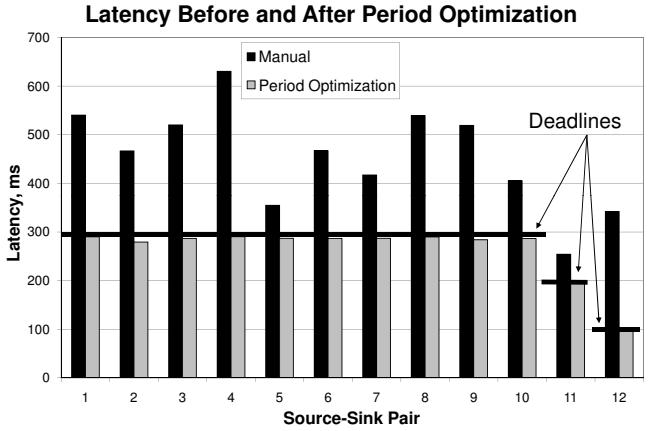

Figure 3: Period optimization meets all deadlines

End-to-end deadlines are placed over paths between 12 pairs of source-sink tasks in the system. Most of the paths follow a six-stage structure: sensor preprocessing & sensory fusion, object detection, selection of target objects in the environment, core functions, vehicle longitudinal & lateral controls with actuator arbitration & planning, and, finally, low-level loops of the actuators themselves. Most of the intermediate stages are shared among the tasks. Therefore, the graph is quite densely connected and despite the small number of source-sink pairs, there are 222 unique paths among them.

The deadline is set at 300 ms for 9 of these source-sink pairs, at 200 ms for two pairs, and at 100 ms for one pair. For 9 pairs of local tasks over 2 ECUs, harmonicity constraints with fixed integer constants are present. Some task and message rates are bounded explicitly, due to controller requirements and maximum sampling rates from sensors. To provide for future extensibility and a safety margin, maximum utilization parameters u_i from (8) are set at 70% for all ECUs and buses.

The system configuration used is a snapshot from an early study of the possible architecture configurations, in which the periods of task and messages had not been finalized. The preliminary manual estimates are based on designer intuition. These initial period assignments, in the worst case, do not meet any of the deadlines as shown in Figure 3.

Starting with all the α parameters equal to 1, we perform a GP optimization. The results of this optimization are also shown in Figure 3. All 222 paths between the 12 source-sink pairs meet their deadlines. The GP problem takes 24 seconds to solve on a 1.6 GHz Pentium M processor with 768 MB of RAM. The GP period assignments are quite different from the manual ones; the average period increases by 90%.

To determine the effectiveness of the iterative procedure, we can track the reduction in $\max(|e_i|), \forall o_i \in O$ across several iterations. The results are shown in Figure 4. 15 iterations of Algorithm 1 are shown on the x-axis. The y-axis (with a logarithmic scale) shows the *maximum* absolute estimation error for the response time estimate used within the GP formulation. The *average* estimation error, not shown, drops from 6.98% to 0.009% during these same 15 iterations. Overall, the maximum estimation error is reduced by a factor of 102, while the average estimation error decreases by a factor of 780. The discrepancy between the approximated ($\sum_{o_i \in \mathcal{O}} s_i$) and actual ($\sum_{o_i \in \mathcal{O}} r_i$) objective values drops from 27.1% during the first iteration to 0.0045% during the final iteration.

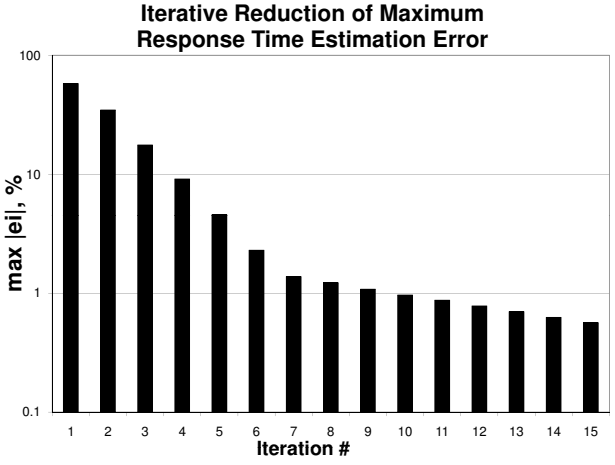

Figure 4: Iterative reduction in maximum estimation error

Since the runtime per iteration is independent of the α values, the total solver time for 15 iterations is 6 minutes. Even though the α values are reduced below 1, (4) and (7) from the eaxct problem are not violated during any of the 15 iterations.

Finally, we can relax the 9 harmonicity constraints from fixed integer constants to integer variables. This changes the problem from a GP to a Mixed Integer GP. The *bnb* solver within Yalmip applies a branch-and-bound procedure to find the solution, and the solution time increases to 227 seconds per iteration.

4.2 Fault-tolerant Distributed System

This system is based on the example given in [17] and contains task replicas allocated to different ECUs for fault tolerance. The system consists of 43 tasks and 36 messages deployed onto an architecture with 8 ECUs and a single bus. The bus is assumed to run at 250kb/s. Initial period assignments for tasks are taken from the example, while initial message periods are assumed to be equal to the source task periods. Task and message priorities are assigned using the rate monotonic rule. The initial end-to-end latencies for six paths in the system are noted.

The experiments for this system are concerned not just with meeting end-to-end delay constraints, but with reducing the path latencies as much as possible while meeting resource utilization bounds. Utilization bounds are set at 70% for each of the 9 resources, and deadlines for the six paths are set to their initial latencies.

First, we attempt to minimize average path latency on the six paths by modifying the objective function. After 15 iterations, each of which takes 1.25 seconds, the average path latency is reduced by 45%. The average utilization for the 8 ECUs is increased from 56% to 61% while the bus utilization is reduced from 74% to 52%. Next, we carry out six more experiments where we minimize each of the individual path latencies separately. The latencies for each of the six paths can be decreased an additional 17% to 63%, for a total reduction ranging between 55% and 70% from the initial latencies.

These experiments demonstrate that it is possible to customize the approach for a modified flow where the designer is interested in minimizing specific path latencies. Even without modifying allocations or priority assignments, period assignment alone is capable of significantly affecting end-to-end latencies in the system.

5. CONCLUSIONS

The continuing proliferation of distributed automotive functionality and architectures complicates the mapping process for these systems. This work provides an optimization procedure that automates the period assignment stage within mapping. First, by leveraging schedulability analysis, we develop an MIGP formulation that is applicable for systems with run-time priority-based scheduling. Next, the MIGP formulation is approximated by a GP formulation and the approximation error between the two formulations is reduced with an iterative procedure. The approach has been applied to two case studies and shown to be efficient, accurate, and extensible. In the future, this work will be integrated with the earlier mapping stages of the design flow shown in Figure 1 in order to carry out joint allocation, priority assignment and period assignment. We are also considering synthesizing hybrid data-driven and periodic activation models [19] for such systems.

6. REFERENCES

[1] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. L. Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91, January 2003.

[2] R. Bosch. CAN specification, version 2.0. Stuttgart, 1991.

[3] S. Boyd, S. Kim, L. Vandenberghe, and A. Hassibi. A tutorial on geometric programming. *Optimization and Engineering*, 2006.

[4] S. Boyd and L. Vandenberghe. Convex optimization. Available at `http://www.stanford.edu/~boyd/cvxbook/`, 2004.

[5] Flexray. Protocol specification v2.1 rev. a. Available at `http://www.flexray.com`, 2006.

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

[7] M. G. Harbour, M. Klein, and J. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Transactions on Software Engineering*, 20(1), January 1994.

[8] B. Kienhuis, E. F. Deprettere, P. van der Wolf, and K. A. Vissers. A methodology to design programmable embedded systems - the Y-chart approach. volume 2268 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2002.

[9] K. Koh, S. Kim, A. Mutapcic, and S. Boyd. gpposy: A matlab solver for geometric programs in posynomial form. Technical report, Stanford University, May 2006.

[10] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The MARS approach. *IEEE Micro*, 9(1):25–40, Feb. 1989.

[11] J. Löfberg. Yalmip : A toolbox for modeling and optimization in MATLAB. In *Proc. of the CACSD Conference*, Taipei, 2004.

[12] M. Di Natale, P. Giusto, S. Kanajan, C. Pinello, and P. Popp. Architecture exploration for time-critical and cost-sensitive distributed systems. In *Proceedings of the SAE Conference*, 2007.

[13] OSEK. OS version 2.2.3 specification. Available at `http://www.osek-vdx.org`, 2006.

[14] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *10th International Symposium on Hardware/Software Codesign (CODES 2002)*, pages 187–192, Estes Park, Colorado, USA, May 6-8 2002.

[15] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *Proceedings of the 11th Real Time and Embedded Technology and Applications Symposium*, pages 160–169, San Francisco (CA), U.S.A., Mar. 2005.

[16] M. Saksena and S. Hong. Resource conscious design of distributed real-time systems – an end-to-end approach. In *Proc. IEEE Int'l Conf on Engineering of Complex Computer Systems*, 1996.

[17] K. 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, 1992.

[18] K. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network CAN message response times. *Control Eng. Practice*, 3(8):1163–1169, 1995.

[19] W. Zheng, M. Di Natale, C. Pinello, P. Giusto, and A. Sangiovanni-Vincentelli. Synthesis of task and message activation models in real-time distributed automotive systems. In *Proc. of Design Automation and Test, Europe*, 2007.