A Survey of Real-Time Automotive Systems[*]

Zhishan Guo, Rui Liu, Xinghao Xu and Kecheng Yang
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

In many cyber-physical systems that support real-time applications, temporal guarantees are crucial. Automotive systems are such an example. In this paper, we survey selected prior work that addresses real-time issues in automotive systems. The covered topics includes real-time analysis of distributed architectures in automotive systems, two component-specific studies on Engine Control Unit and Adaptive Cruise Control, and scheduling analysis of adaptive varying-rate tasks.

1 Introduction

For safety critical cyber-physical systems such as automotive systems, absolute guarantee of timely behaviors are crucial. In fact, an extensive amount of work in the real-time system research community have been motivated by applications in automotive systems. This paper aims to provide a report of the current trends in real-time automotive systems research.

In modern automotive systems that support real-time applications, distributed architectures are commonly used, where several buses (*e.g.*, CANs) and tens of computation units (*e.g.*, ECUs) are connected in a complex network system. To ensure a system-wide timing guarantee, crucial components must be studied in the first place. Section 2 provides an introduction to the CAN message protocol and existing schedulability results. Moreover, to guarantee the timely performance of the real-time workloads in such systems, end-to-end latencies for those workloads are critical. Section 3 considers two activation models that provide different balances between latencies and schedulability.

Section 4 looks at two concrete examples of how real-time scheduling techniques are used to improve performance and safety for automotive systems. One proposes a dynamic scheduler for Engine Control Unit to make CPU used with higher efficiency. The other one focuses on real-time support on Adaptive

[*]Comp 790-062 Automotive Cyber-Physical Systems Course Project Paper

Cruise Control System, including mode change technique, server technique, static priority scheduling and background scheduling in real-time domain.

Section 5 studies the usage of adaptive varying-rate (AVR) task model. The AVR task model has been proposed as a means of modeling certain physically-derived constraints in cyber-physical systems (CPS) in a manner that is more accurate (less pessimistic) than is possible using prior task models from real-time scheduling theory.

The pieces of work included in this paper are part of the efforts that seek to define and better understand the role that real-time scheduling theory should play within the emergent discipline of cyber-physical systems, especially automotive ones.

2 Real-Time Scheduling on CAN

Controller Area Network (CAN) is a serial data communication bus that is commonly used in automotive systems nowadays. It was first developed by Robert Bosch GmbH in 1983. Intel and Philips manufactured the first CAN controller chip in 1987, followed by the first ISO standard in 1993. Among automotive companies, Mercedes was the first to deploy CAN in an actual production vehicle (the 1991 S-Class). In the mid 1990s, as the number of ECUs in automotive systems grew rapidly, the traditional point-to-point wiring became hard to manage and costly due to its huge demand for copper wire. Due to its relatively simple physical structure, CAN started to be adopted as the main solution to the increase in on-board electronics contents. By 2004, there were over 50 different microprocessor families with on-chip CAN capability, and CAN has become a mandatory equipment for all cars and light trucks. For the remainder of this section, we first give a short introduction to the CAN message protocol with a focus on real-time scheduling and then discuss the scheduling model and related schedulability results.

2.1 CAN Message Protocol

Nodes on a CAN network communicate with each other through 4 types of frames: data frame, remote transmit request frame, overload frame, and error frame. The scheduling mainly focuses on data frames. A typical layout of a data frame is given in Figure 1. The majority of a data frame is the *data field* which has variable size of $8N (0 \leq N \leq 8)$ bits. Normally, CAN nodes are only allowed to start transmission when the bus is idle. To resolve potential contention due to the lack of a global synchronized clock, data

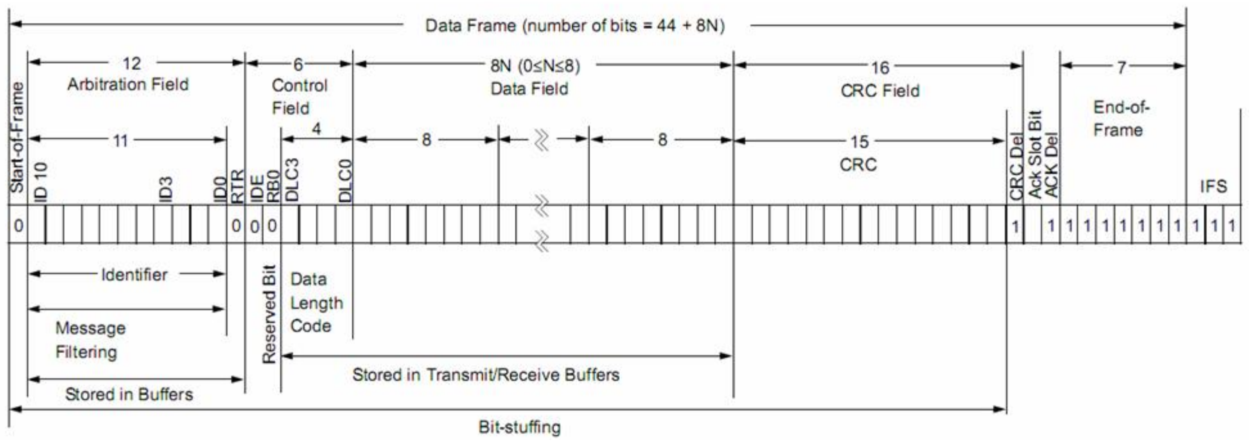


Figure 1: CAN data frame

transmission on CAN relies on a lossless bit-wise *arbitration process*. This arbitration process requires all nodes on the CAN network to be synchronized to sample every bit on the CAN network at the same time. Such synchronization process is initiated by the *start-of-frame* bit. Following the start-of-frame bit is the 11-bit frame identifier, which also indicates the priority of this frame. Identifier is part of *arbitration field* which, as its name suggests, is used in the arbitration process. For more details of the arbitration process, please refer to [6] for a complete description.

Error Detection and Bit Stuffing Following the data field, these is a 15-bit field called *Cyclic Redundancy Check (CRC)* which is used to check errors in transmitted messages. When an error is detected by a node, this node sends an 6-bit error flag (“000000” or “111111”). Once an error flag is sent by one node, all other nodes will be informed of the detected error.

Since bit patterns “000000” and “111111” have special meanings in error detection, there needs to be a way to distinguish them from the variable part of the transmitted messages. The *bit-stuffing* method is proposed to resolve such confusion. In the bit transmission process, a bit of the opposite polarity is inserted after 5 consecutive bits of same polarity are transmitted. A worst-case bit stuffing scenario is depicted in Figure 2.

2.2 Scheduling Model

The hardware platform under consideration comprises a number of nodes (microprocessors) connected via CAN. The task system is assumed to contain a static set of hard real-time messages, and each message m

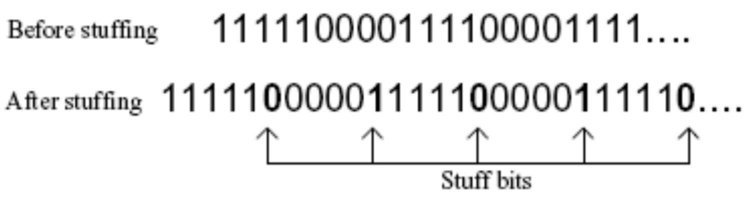

Before stuffing 111110000111100001111....
After stuffing 11111000001111100000111110....
Stuff bits

Figure 2: A wost-case bit-stuffing as depicted in [12]. A opposite-polarity bit is inserted after every 4 (initially 5) bits.

is statically assigned to one node. Moreover, each message m has a fixed identifier. This identifier includes complete information about the message priority. Thus, each message m has an unique priority. At the start of the transmission process, each node attempts to transmit the highest priority message queued at that node. Once transmitted, the messages are scheduled non-preemptively.

Message m also has a maximum number of data bytes s_m and a maximum transmission time C_m. Each message is assumed to be queued by a software task, which is triggered by events. This task needs a bounded amount of time between 0 and J_m to complete the queuing of a message. J_m is referred to as *queuing jitter*. The event that triggers the queuing of message m is assumed to occur with a minimum separation time of T_m, which is referred to as the *period*.

Each message m also has a hard *deadline* D_m, corresponding to the maximum permitted time from the occurrence of the triggering event to the end of the successful transmission of the message (i.e. all node requesting m have received it). Receiving nodes may have different timing requirements, and we assume D_m to be the tightest of those. We define the worst-case response time R_m to be the worst-case time between the occurrence of the triggering event and the end of message transmission. Thus, message m is schedulable if and only if $R_m \leq D_m$.

2.3 Existing Results

Under the model described above, Tindell and Burns [25] and Tindell et al. [26] showed how to compute worst-case response times for all messages. Tindell's research was very influential in both academia and industry: it has led to a large body of related work (over 200 papers), and it has influenced the design of Motorola msCAN [12]. Prior to this work, bus utilization had a typical level of 30% to 40%, with extensive testing to ensure that deadlines are met. However, with the systematic analysis proposed by Tindell, CAN bus utilization could reach 80% without exhaustive testing.

However, it was pointed out by Davis et al. [12] that Tindell's analysis could potentially compute worst-case response times that are optimistic. Davis et al. proposed a new analysis that removes the optimism in analysis by leveraging existing results on fixed priority preemptive scheduling with arbitrary deadlines and on fixed priority non-preemptive scheduling [12]. At first it would be a surprise that results on preemptive scheduling are used here, since the scheduling model requires message scheduling to be non-preemptive. However, non-preemptive scheduling can be viewed as a special case of preemptive scheduling: a job's priority is adjusted to the highest priority before execution to prevent preemption. Davis et al. also showed that the optimal priority assignment by Audsley [1] is applicable to CAN messages.

2.4 Modeling Faults

One of the hard challenges in the study of a message passing system is how to deal with faults. In the case of CAN, we are mainly concerned with communication faults. Although the CAN message protocol provides a robust mechanism to detect (through CRC) and recover from faults, such a mechanism introduces nontrivial recovery cost and communication delays that must be incorporated into the schedulability analysis. The initial assumption about CAN faults is that there exists a minimum separation time between faults. However, this assumption is not realistic. A major cause of communication and network faults in CAN is electromagnetic interference. Under such circumstances, a minimum separation time may not exist. A more general approach to model faults is by counting the number of errors that have occurred up to time t through a function $F(t)$. Broster et al. [7] considered the use of Poisson Process to model faults, that is $F(t)$ is now a Poisson random variable w.r.t. time. The Poisson process has been used extensively to model occurrence of random events in various disciplines due to its well-founded theoretical study and some ideal characteristics (*e.g.* Memoryless-property and uniform distribution over time). After incorporating Poisson faults into the model, Broster's analysis yields a distribution of the worst-case response time for each message. One thing to note is that Broster's analysis is still based on the flawed schedulability analysis by Tindell. Nevertheless, Davis et al. claimed that the their revised analysis can support various fault models including the Poisson fault model [12].

3 Real-Time Distributed Automotive Systems

In prior work on distributed automotive systems, two activation models, the periodic activation model and the data-driven activation model, are applied to provide end-to-end latencies guarantee for real-time work-

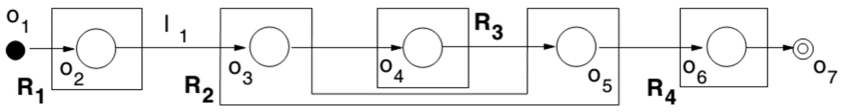


Figure 3: Illustrative example for the system model.

loads. [19] and [27] studied hybrid approaches of the two activation models to meet certain temporal requirements. On the other hand, [10] assumes the periodic activation model for the whole system, and optimizes the period assignment to provide timely guarantees.

3.1 Model

In [10, 19, 27], the workload model is a *dataflow*, which is represented by a *Directed Acyclic Graph.* This model is defined by a tuple $\{\mathcal{V}, \mathcal{E}, \mathcal{R}, \}$, where $\mathcal{V}$, $\mathcal{E}$, and $\mathcal{R}$ are the sets of vertices, edges, and resources, respectively. Each resource $R_z \in \mathcal{R}$ either supports the execution of tasks (CPUs, ECUs) or the transmission of messages (buses, CANs).

$\mathcal{V} = \{o_1, \cdots, o_n\}$ is the set of *objects*, which is either a task or a message and is characterized by (C_i, R_i), where C_i is the worst-case time needed for o_i to complete execution or transmission and R_i is the resource on which o_i executes or transmits. All objects are scheduled by their static priorities. The lower the index, the higher the priority. $r_i$is the worst case response time of o_i, from the activation of the object to its completion in case it is a task, or its arrival at the destination node in case it is a message. w_i is defined as the worst case time spent from the instant the job is released with maximum jitter J_i to its completion or arrival, *i.e.*, $r_i = J_i + w_i$. Each object o_i has a *period* T_i.

The set of edges $\mathcal{E} = \{l_1, \cdots, l_m\}$ is the set of links. A link $l_i = (o_h, o_k)$ connects the output port of o_h (the source $src(l_i)$) to the input port of o_k (the sink $snk(l_i)$).

$P_{i,j}$ denotes a *path* from o_i to o_j, which is an ordered sequence $P = [l_{q_1}, ..., l_{_k}]$ of links that starts from $o_i = src(l_{q_1})$ and ends at $o_j = src(l_{q_k})$. Then, o_i (o_j) is the source (sink) of the path. The worst-case end-to-end latency of $P_{i,j}$ is denoted $L_{i,j}$.

Figure 3 is an illustrative example for this model. o_1 (o_7) represents a "null node", which is no task nor message and with no workload, for the starting (complete) point of the flow. o_2, o_4, o_6 could be three tasks which need to be scheduled on three different ECUs R_1, R_3, R_4, respectively. o_3, o_5 are messages that are transmitted by the CAN bus R_2.

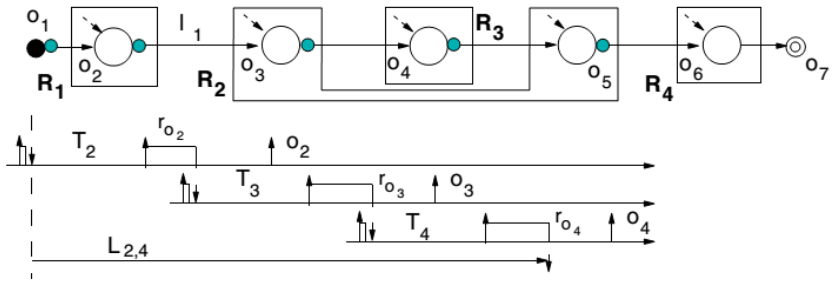


Figure 4: Illustrative example for the periodic activation model.

There are two common *activation models* for the activation of the objects in a real-time distributed automotive system, namely the *periodic activation model* and the *data-driven activation model*.

Periodic activation model. In this model, every task is activated according to the freshest messages that are periodically received from an asynchronous buffer, and every message is triggered periodically according to the latest received signals. Therefore, in the periodic activation model, every object has zero release jitter. Moreover, for each path, the worst case end-to-end latency is the sum of the worst-case response times and period of all objects in this path. That is,

$$L_{i,j} = \sum_{o_k \in P_{i,j}} (T_k + r_k).$$

Figure 4 is an illustration for the periodic activation model. Since in the periodic activation model each object can only be activated periodically, the worst-case end-to-end latency happens when every complete signal arrives right after a periodic activation "tick". That is why we need to sum all periods of objects in the path.

Data-driven activation model. In this model, for intermediate nodes in a computation chain, *i.e.*, $o_i \to o_j$, the maximum release jitter for the latter is $J_j = r_i$. Thus, the worst case end-to-end latency can be computed by summing all w_i in this computation chain. That is,

$$L_{i,j} = \sum_{o_k \in P_{i,j}} w_k.$$

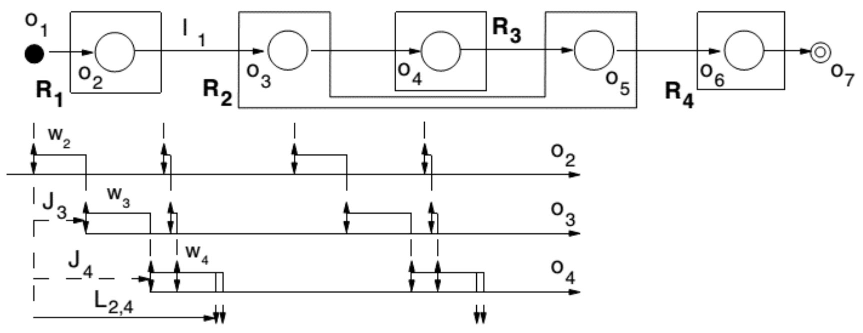


Figure 5: Illustrative example for the data-driven activation model.

Figure 5 is an illustration for the data-driven activation model. Every object is released immediately after its predecessor in the computation chain completes.

Generally, under the periodic activation model, even high-priority computation chains could have significantly high end-to-end latencies while low-priority ones have a possible better schedulability; on the other hand, under the data-driven activation model, high-priority computation chains have low end-to-end latencies while low-priority ones may suffer from very high end-to-end latency due to bursty activations of high-priority objects.

3.2 Hybrid Activation Model

Due to the pros and cons of the two activation models as discussed above, both [19] and [27] consider a hybrid of the two activation model to improve the end-to-end latency analysis. That is, they let a subset of tasks and messages to be *periodically* activated while letting the remaining ones be *data-driven* activated.

In [19], a *Search Algorithm* is proposed. It starts with the assumption that all objects are periodically activated, and then systematically makes an object data-driven activated. For each change, new end-to-end latencies are computed and a *critical-path* is determined. If the latencies still do not satisfy the system requirements, the it goes to next stage until the requirements are satisfied. Figure 6 describes the search tree of the Search Algorithm, which generally searches activation configurations of the system from periodic activation towards data-driven activation.

In [27], a solution of *mixed-integer linear programming* (MILP) is proposed. The MILP consists of a set of linear constraints, where a subset of variables have to be integers. Furthermore, different object functions can be applied for different optimization purpose. In [27], four example object functions are considered.

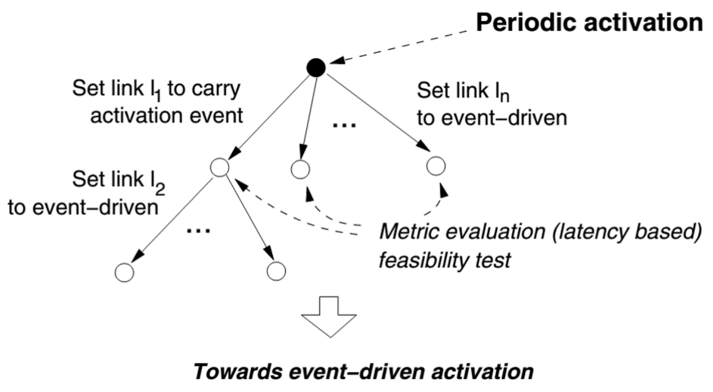
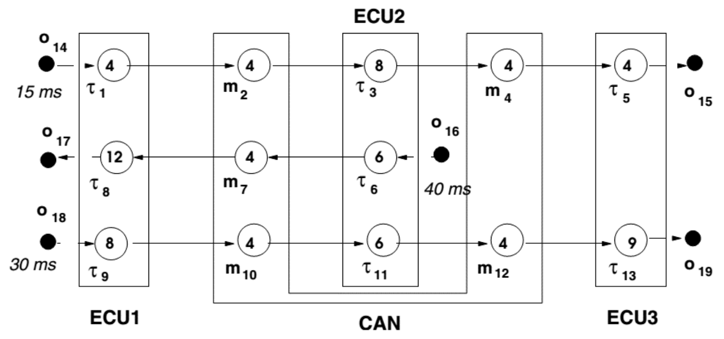

Periodic activation
Set link l1 to carry activation event
Set link ln to event–driven
Set link l2 to event–driven
...
Metric evaluation (latency based) feasibility test
Towards event–driven activation

Figure 6: The search tree for the algorithm in [19].



				periodic		*event − driven*		
Object	π_i	T_i	C_i	r_i	L_i	J_i	w_i	r_i
τ_1	13	15	4	4	4	0	4	4
m_2	12	15	4	8	27	4	8	12
τ_3	11	15	8	8	50	12	8	20
m_4	10	15	4	12	77	20	12	32
τ_5	9	15	4	4	**96**	32	4	**36**
τ_6	8	40	6	14	14	0	30	30
m_7	7	40	4	16	70	30	28	58
τ_8	6	40	12	20	**130**	58	20	**78**
τ_9	5	30	8	28	28	0	60	60
m_{10}	4	30	4	28	86	60	44	104
τ_{11}	3	30	6	28	144	104	58	162
m_{12}	2	30	4	28	202	162	88	250
τ_{13}	1	30	9	13	**245**	250	25	**275**

Figure 7: Example system for the case study in [27].

- F_1 = minimization of the number of event buffers
- F_2 = minimization of the sum of the path latencies
- F_3 = minimization of the sum of weighted lateness for all the paths exceeding the deadline
- F_4 = minimization of the lowest priority path latency

Due to the complexity of MILP, an approximation method is used for solve the MILP. A case study is provided to illustrate this MILP approach. Figure 7 shows the system for the case study, where the table provides the parameters of the system and analytical results under either purely periodic or purely data-driven activation model. In either model, the end-to-end latencies for the three computation chains are highlighted in bold.

As seen in the table in Figure 7, if the deadlines of the three computation chains are 80, 120, 260, then

Objective	P_1	P_2	P_3	periodic objects	event objects
F_1	51	58	233	m_3	remainings
F_2	66	58	191	τ_3, m_4, τ_{10}	remainings
F_3	51	110	209	m_2, m_7	remainings
F_4	66	98	157	τ_3, m_4, τ_8	remainings

Figure 8: Results for the case study in [27], applying the MILP.

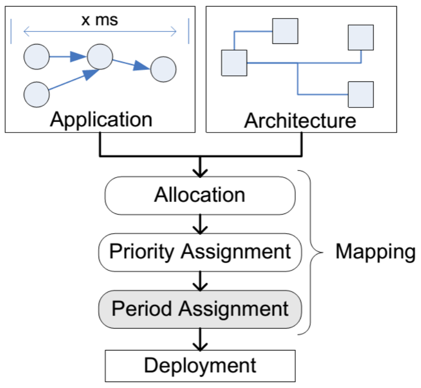


Figure 9: The design flow in [10].

neither of the two pure activation models can satisfy all deadline requirements. However, if applying the MILP algorithm, the resulting latencies and the subset of objects that are periodically activated are provided in Figure 8. As shown, using any of the four object functions described earlier, it results in all of the three computation chains meeting their deadlines of 80, 120, 260, respectively.

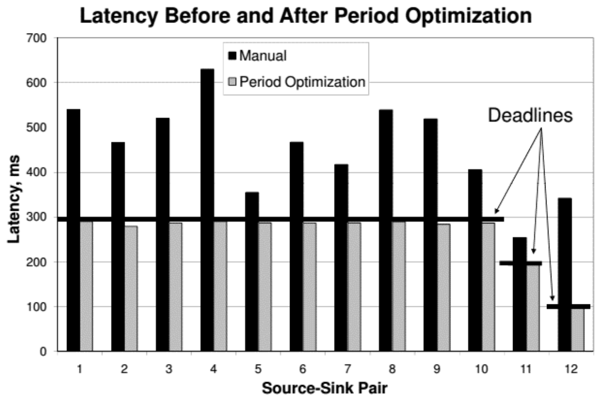
3.3 Period Optimization

Instead of improving the analysis by combining two activation models, [10] assumes the periodic activation model for all objects. According to a system design flow as described in Figure 9, [10] focuses on optimizing the period assignment.

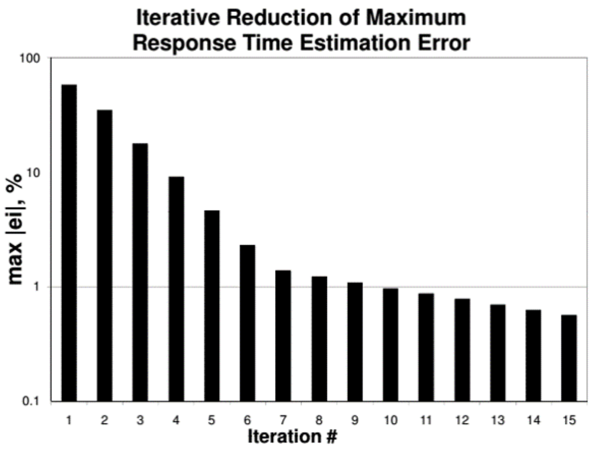
In [10], such a optimization is done by *mixed-integer geometric programming* (MIGP). A *geometric program* (GP) consists of a set of posynomial inequalities and a set of monomial equalities. A posynomial is a sum of one or more monomials. A monomial is in the following form:

$$m(x) = c x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}, \quad \text{where } c > 0 \text{ and } a_i \in \mathbb{R}.$$

If some variables in a GP have to be integers, then it becomes a MIGP. Unlike GP, MIGP cannot be



(a) Period optimization results.



(b) Estimation errors accross iterations.

Figure 10: Case study in [10].

efficiently solved, therefore [10] uses an approximation approach to solve it approximately. Also, [10] approximately computes the solution for the MIGP iteratively. After each iteration, a notion of *error* is computed, which is used to set up next iteration. The lower the error, the more precise the approximation result.

To illustrate this period optimization approach, [10] also conducted a case study. As shown in Figure 10(a), the manually set periods, which is barely based on the intuition of the designer, cannot guarantee deadlines are met in the worst case, while the period optimization can provide a solution where all deadlines are guaranteed to be met. Such timely guarantees are at the cost that the average period is increased by 90%. Also, as shown in Figure 10(b), the estimation error is reduced to less than 1% after 10 iterations.

4 Application

Multicore and multiprocessor real-time systems are increasingly used in automotive systems to provide advanced safety control and offer a comfortable experience for customers. Automotive Open Systems Architecture (AUTOSAR) [2] suggests static priority scheduling. However, there are two limitations of such an implementation. One is the lack of guarantee for lower priority timing constraints. The other one is that the resource sharing problem cannot be solved very well. In order to improve these limitations, Mishra and Gurumurth [17] propose a dynamic scheduling algorithm on automotive systems. They also provide a case study on Engine Control Unit. The real-time task is defined with three parameters: period, worst case

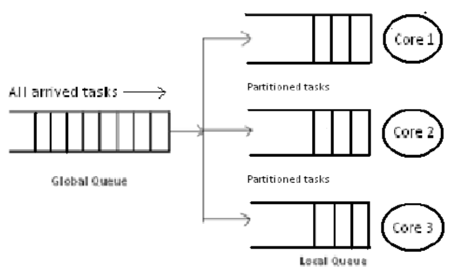


Figure 11: Dynamic scheduler structure

execution time (WCET) and deadline. The structure of the dynamic scheduler is shown in Figure 11.

There is one global queue which is used to sort all arrived tasks. Each core has a partitioned queue owned by itself. At the global queue, the slack of each task will be calculated. Slack here is defined as (period minus WCET). Each task will be assigned a priority according to their slack. The highest priority will be given to the task with least slack. Tasks with highest priority in the global queue will be dynamically allocated to a specific core on which it will be executed. The assignment decision can be made based on three cases. If there is a precedence constraint between the current task and some previous tasks, then the current task will be assigned to the same core where the previous task is allocated. If there is no precedence constraint relationship and some cores are available to use, then the current task can select one of them for execution. Otherwise, the current task will be assigned to the core whose total WCET of all assigned tasks is the least.

After tasks are dynamically assigned to each partitioned core, they will be scheduled on multicores according to the rules as follows: if the task assigned to some partitioned queues cannot be executed immediately, we have to compare the slack of the current task and its previous task to determine if additional operations are required.

- Swap operation will be performed if the following two conditions hold:

    1. Slack of new arrival $<$ WCET of previous task $+$ Remaining WCET of running task
    2. Slack of previous task $>$ WCET of new arrival

- Migration operation will be performed if the following two conditions hold:

    1. Slack of new arrival $<$ WCET of previous task $+$ Remaining WCET of running task
    2. Slack of previous task $<$ WCET of new arrival

Scheduling Method	CPU Utilization	Deadline Miss	Average Response Time	Preemption	Migration
Static	100% by higher priority tasks	30% for 1st instance	4 units	30%	20%
Dynamic	100% by fair allocation	3% for 3rd instance	1.6 units	10%	10%

Table 1: Simulation result with static scheduling and dynamic scheduling

Simulation results are presented in Table 1. The advantage of dynamic scheduling is easily observed. In static scheduling, CPU is highly utilized by high priority tasks and deadlines of low priority tasks are frequently missed. However, this shortcoming can be overcome by using dynamic scheduling. In addition, compared to static scheduling, dynamic scheduling has lower average response time and overhead, including preemption and migration.

Goud et al. [13] presented real-time support for Adaptive Cruise Control System (ACCS). ACCS provides an intelligent way to adjust vehicle speed to guarantee safety propriety. ACCS basically performs two main functions. On the one hand, it maintains safety distance if the vehicle ahead is not that far. On the other hand, it keeps to the speed set by the driver if there is no traffic ahead or the distance between two vehicles is far enough. In order to achieve the above two functions, ACCS uses two level controllers. The Upper-level controller is responsible for computation. The Lower-level controller is responsible for manipulating hardware to achieve the desired speed. Two kinds of real-time data in ACCS, raw data and derived data, are involved in the calculation of desired speed. Raw data is used to reflect the state of the external environment and is achieved by sensors or radars. Derived data is calculated from raw data and other derived data and can be used by specific tasks as instructions or criteria. The real-time framework in ACCS is shown in Figure 12. The circle in the figure represents raw data or derived data. The arrows in the figure stand for different kinds of real-time tasks. Goud et al. presented several real-time support based on the data and tasks shown in this framework.

Real-time supports in ACCS are mainly focused on two aspects, mode change technique and real-time data updates. There are two real-time modes in ACCS, Non-Critical Mode (NC Mode) and Safety-Critical Mode (SC Mode). They are mutually exclusive and the vehicle is in one mode at any time instant while running. In SC Mode the parameters of the system are rapidly changing while in NC Mode they are not. The mode selection depends on two parameters, Distance of Separation (DoS) and Rate of change of Distance (RoD). All possible conditions for these two modes are listed in Table 2.

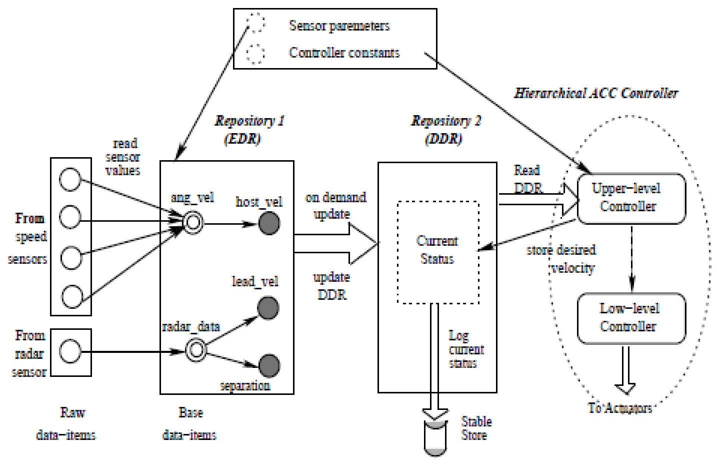


Figure 12: Real-time framework in ACCS

LeadDist	RoD	mode
FAR	Decr-Fast	SC
FAR	Incr-Fast	NC
FAR	Decr-Slow	NC
FAR	Incr-Slow	NC
NEAR	-	SC
FOLLOW	-	Retain Mode

Table 2: Possible conditions for mode change

Mode change in real-time systems is a very common and useful technique. In ACCS, mode change usually includes adding tasks to the task set, removing tasks from the task set or changing the parameters of specific tasks. All these changes should maintain the scheduability. More details about mode change protocols in real-time systems can be found in [22]. In automotive systems, dual mode is designed to achieve efficient utilization of CPU while maintaining safety propriety. Mode change techniques guarantee the timing constraints of real-time tasks. Once the mode is determined, all the parameters of tasks are known a priori. Rate Monotonic Algorithm will be selected to schedule the tasks executing in different modes.

There are two data repositories in ACCS. One is called Environment Data Repository (EDR) and the other one is called Derived Data Repository (DDR). Real-time data (raw data and derived data) is stored in these two repositories and processed by different kinds of tasks. Goud et al. specified four different kinds of tasks and presented real-time techniques to support them.

- Sensor Reading Task

  Sensors detect the parameters of the external environment. A Sensor Reading Task is responsible for reading those parameters and transferring them into EDR. A Sensor Reading Task can be considered as a periodic task and will be scheduled by a static priority scheduler.

- On-demand Update Task

  On-demand Update Tasks are responsible for calculating raw data into derived data and updating derived data if necessary. Derived data will be updated only if one input parameter changes more than the threshold value. Because of this propriety, an On-demand Update Task is modeled as an aperiodic task. In order to ensure the timing constraints, an aperiodic server technique is applied for On-demand Update Tasks. Except for regular server which reserves a share of processor bandwidth, there is a controlling server in ACCS to schedule maintain each reserved bandwidth.

- Lower Level Controller Task

  Lower Level Controller Tasks are responsible for manipulating mechanical systems to achieve desired speed. They are usually performed by hardware.

- Other Tasks

  Other Tasks include lower priority tasks which are responsible for weather monitoring and road scanning. Because of their lower priority, these tasks are scheduled in the background, which is known as background scheduling in real-time domain.

5 Scheduling AVR Tasks in Automotive Systems[1]

5.1 Background and Introduction

Rigorous model-based design (MBD) approaches are widely used in the development of automotive systems. Unfortunately, there is not always a perfect fit between the models of MBD and the lower-level ones of scheduling theory, and consequently the mapping is achieved by introducing additional *pessimism*. Such pessimism in the mapping process results in resource under-utilization during run-time. In an attempt to reduce such implementation inefficiency and thereby be able to obtain more resource-efficient implementations of cyber-physical systems, real-time scheduling theory has recently begun looking at developing, from first principles, new low-level *task models that are inspired by requirements of actual physical systems*.

[1]This section of the report is modified from part of [14] under permission.

```
task sample_task {
  rpm = read_speed();
  f1();
  if(rpm < 4000) {
    f2();
  }
  if(rpm < 2000) {
    f3();
  }
}
```

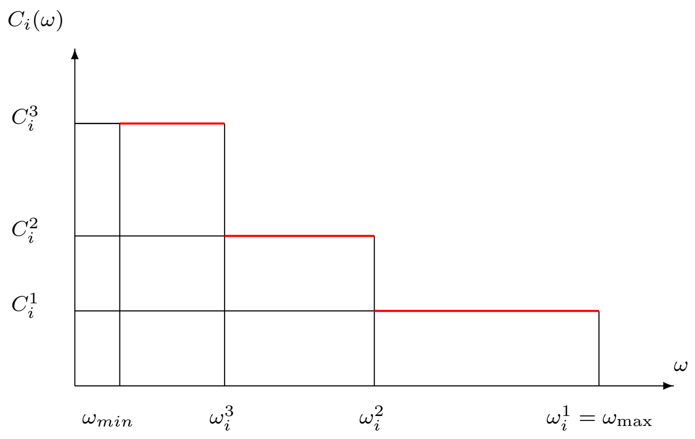


Figure 13: An AVR task with WCET dependent to the engine rotation speed.

All the research work that we describe in this section deals with the modeling of recurrent processes in cyber-physical systems for which each activation of the recurrent process is triggered by the values of variables describing the state of the physical system. Such processes abound in cyber-physical systems: for example, height detection in avionic systems is activated more frequently at lower altitudes; sensor acquisition in mobile robots often depends on the robot location (e.g., whether it is approaching edge areas); while fuel injection in the Engine Control Unit (ECU) of an automobile is dependent upon the position of each piston (which is a function of the crankshaft angular position).

5.2 System Model

Consider the ECU, a typical hard real-time system, as an example, on which some tasks execute at a varying rate depending upon engine speed. These tasks calculate the quantity of fuel to be injected, and the precise instants at which the injected fuel is to combusted, in order to achieve optimal engine performance (maximize the thrust obtained, avoid engine knocking, etc.). A key challenge in the schedulability analysis of collections containing one or more such tasks is that not only the periods and deadlines, but also the execution requirements, are determined by specific angular positions of the crankshaft or its rotation speeds, which relate to the engine speed [9]. Since engine behavior is generally more stable when running at higher speeds, some functions (that must execute at lower speeds) do not need to execute at higher speeds. Figure 13 shows a typical piece of pseudocode for a task with worst-case execution time (WCET) dependent on crankshaft rotation frequency, and further illustrates such a relationship under a WCET-speed function.

As depicted in Figure 13, an AVR task $\tilde{\tau}_i$ is composed of M_i *modes*: $\mathcal{M}_i = \{(C_i^m, \omega_i^m), m = 1, 2, ..., M_i\}$, where $\omega_i^{M_i+1} = \omega_{\min}$, and $\omega_i^1 = \omega_{\max}$. As engine speed falls in the range $[\omega_i^m, \omega_i^{m+1})$, mode $\mathcal{M}_i^m$ is triggered when job activation occurs, and the AVR job's WCET is C_i^m.

The challenge with this type of task is that the time between successive activations is neither constant nor arbitrary; rather, it depends on the engine rotation speed, which varies within a specified range with specified maximum acceleration and deceleration. If we were to use classical real-time scheduling analysis (for example, the sporadic model [16, 18]), the relationship between activation period and WCET can only be modeled by introducing unnecessary and excessive pessimism.

Consider a set of n tasks to be scheduled on a preemptive uniprocessor. Each task τ_i generates an infinite sequence of jobs $J_{i,1}, J_{i,2}, ...$, and can either be a *regular sporadic* task, characterized by a (fixed) WCET c_i, period T_i, and relative deadline D_i, or an *adaptive varying-rate (AVR)* task, where all three parameters are variables.

The activation pattern and functionality of an AVR task $\tilde{\tau}_i$ are determined by the physical evolution of the engine. Specifically, each subsequent job $J_{i,k}$ is released when the crankshaft reaches a predefined angular position within each *revolution*, and the inter-arrival time between two consecutive jobs $J_{i,k}$ and $J_{i,k+1}$ is denoted as the activation period $T_{i,k}$, which is a function of the crankshaft rotation speed ω of the engine.

5.3 Existing Work

This problem, formalized as the Adaptive Varying-Rate (AVR) tasks scheduling problem, was first introduced to the real-time system community in a keynote address at ECRTS [9] in 2012, where the schedulability analysis challenges of such tasks are highlighted.

Early Work. Some early work on understanding the scheduling of AVR tasks was done by [15] on a simplified model, where only a *single* task, that is assigned highest priority, is analyzed. Pollex et al. [21] later considered systems with multiple tasks, but constant engine speed, and do not account for mode transitions during execution. This work was later extended to the case with angular acceleration in [20] — by considering the maximum execution time and the minimum inter-arrival time starting from different speeds, sufficient schedulability conditions were derived.

Fixed-Priority. For the preemptive uniprocessor fixed-priority scheduling of AVR tasks, Davis et al. [11] present an Integer Linear Programming (ILP) based sufficient dynamic schedulability test; Biondi et al. [4] have recently proposed a search tree based method based on a Brute-Force approach with pruning rules, and show its domination over the existing ILP-based one. Exact analysis is later on presented to the worst-case interference generated by a AVR task under fixed priority and arbitrary deadlines [5].

Dynamic-Priority. To the best of our knowledge, [8] is the first piece of work to study the preemptive uniprocessor dynamic-priority scheduling of systems of AVR tasks. [8] and [4] derive a sufficient, but not necessary, schedulability condition under the added simplifying (but not necessarily valid) assumption that when an AVR task switches to a new mode due to a positive acceleration, the next job will run with the computation time associated with the new mode. Very recently, [14] provides the speedup bound analysis for such sufficient utilization based test, which shows that under current physical limitations, the schedulability test is already very close to optimal, and there are little room for improvement. [3] further verifies such theory by experimentally comparing such method with their newly proposed tree-pruning method for schedulability analysis of EDF.

Constrained-Deadlines. Constrained deadlines frequently arise in engine triggered tasks as well. [14] is the first work that seriously tackle this problem, by transforming it into the digraph based task model [23] [24]. [3] further improve the transformation to an exact one, so that there is no capacity lost during the process.

References

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

[2] GbR AUTOSAR. Autosar–technical overview v2. 0.1, 2006.

[3] A. Biondi, G. Buttazzo, and S. Simoncelli. Feasibility analysis of engine control tasks under edf scheduling. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS'15)*, 2015.

[4] A. Biondi, A. Melani, M. Marinoni, M. Di Natale, and G. Buttazzo. Exact analysis of adaptive variable-rate tasks under fixed-priority scheduling. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS'14)*, 2014.

[5] A. Biondi, M. D. Natale, and G. Buttazzo. Response-time analysis for real-time tasks in engine con-

trol applications. In *Proceedings of the 6th International Conference on Cyber-Physical Systems (IC-CPS'15)*, 2015.

[6] R. Bosch. Can specification, version 2.0, Stuttgart, 1991.

[7] I. Broster, A. Burns, and G. Rodriguez-Navas. Probabilistic analysis of can with faults. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 269–278, 2002.

[8] G. Buttazzo, E. Bini, and D. Buttle. Rate-adaptive tasks: Model, analysis, and design issues. In *Proceedings of the International Conference on Design, Automation and Test in Europe (DATE'14)*, 2014.

[9] D. Buttle. Real-time in the prime-time. In *Keynote speech given at the 24th Euromicro Conference on Real-Time Systems (ECRTS'12)*, 2012.

[10] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. S. Vincentelli. Period optimization for hard real-time distributed automotive systems. In *44th DAC*, 2007.

[11] R. I. Davis, T. Feld, V. Pollex, and F. Slomka. Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'14)*, 2014.

[12] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[13] G.R. Goud, N. Sharma, K. Ramamritham, and S. Malewar. Efficient real-time support for automotive applications: A case study. In *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on*, pages 335–341, 2006.

[14] Z. Guo and S. Baruah. Uniprocessor edf scheduling of avr task systems. In *Proceedings of the 6th International Conference on Cyber-Physical Systems (ICCPS'15)*, 2015.

[15] J. Kim, K. Lakshmanan, and R. Rajkumar. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In *Proceedings of the 3rd IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS'12)*, pages 28–38, 2012.

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

[17] Geetishree Mishra and KS Gurumurthy. Dynamic task scheduling on multicore automotive ecus. *International Journal of VLSI design & Communication Systems (VLSICS)*, 5(6), 2009.

[18] A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Massachusetts Institute of Technology, 1983.

[19] M. Di Natale, W. Zheng, C. Pinello, P. Giusto, and A. S. Vincentelli. Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems. In *13th RTAS*, 2007.

[20] V. Pollex, T. Feld, F. Slomka, U. Margull, R. Mader, and G. Wirrer. Sufficient real-time analysis for an engine control unit. In *Proceedings of the 21st International Conference on Real-Time and Networked Systems (RTNS'13)*, pages 247–254, 2013.

[21] V. Pollex, T. Feld, F. Slomka, U. Margull, R. Mader, and G. Wirrer. Sufficient real-time analysis for an engine control unit with constant angular velocities. In *Proceedings of the International Conference on Design, Automation and Test in Europe (DATE'13)*, pages 1335–1338, 2013.

[22] Jorge Real and Alfons Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst.*, 26(2):161–197, March 2004.

[23] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The digraph real-time task model. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'11)*, pages 71–80, 2011.

[24] M. Stigge, P. Ekberg, N. Guan, and W. Yi. On the tractability of digraph-based task models. In *Proceedings of the 22th Euromicro Conference on Real-Time Systems (ECRTS'11)*, pages 162–171, 2011.

[25] Ken Tindell and Alan Burns. Guaranteeing message latencies on control area network (can).

[26] 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):133–151, 1994.

[27] W. Zheng, M. Di Natale, C. Pinello, P. Giusto, and A. S. Vincentelli. Synthesis of task and message activation models in real-time distributed automotive systems. In *10th DATE*, 2007.