Ensuring data freshness for periodic real-time tasks

Evariste Ntaryamira, Cristian Maxim, Liliana Cucu-Grosjean

Ensuring data freshness for periodic real-time tasks

Evariste NTARYAMIRA
INRIA Paris, Paris - France
evariste.ntaryamira@inria.fr

Cristian MAXIM
INRIA Paris, Paris - France
cristian.maxim@inria.fr

Liliana Cucu-Grosjean
INRIA Paris, Paris - France
liliana.cucu@inria.fr

ABSTRACT

Automotive systems are composed of embedded applications which are continuously exchanging real-time data. Exchanged data are then propagated within a list of applications involved at different rates in the definition of a function. Different rates of executions for the applications provoke over- and/or under-sampling of data and the age of the data has an obvious impact on the driving decisions. Ensuring that the appropriate data are consumed by an application motivates to maintain the *freshness* or the *temporal validity* of real-time data. In this paper we propose a method calculating the freshness of the data for real-time systems where multi-rate sampling of data is considered.

1. INTRODUCTION

Automotive systems are composed of embedded applications which are continuously exchanging real-time data, often between applications that are executed at different rates or periods. These different periods of applications, out of which some are producers and others are readers, provoke obvious under- or over-sampling of the data due to the utilization of limited size buffers. The *data age* is a notion introduced to study the impact of freshness of data on the re-activity of the system. Here we define the data age as the time elapsed from the instant a data is stacked into the buffer until the time this data is consumed by the last reader in the chain. The utilization of the data by a chain of applications is seen as the propagation of the data within the system. Different papers are calculating the *maximum data age* in such chains of applications. For instance, end-to-end timing analyses for component-based model systems are provided in [3, 2, 4]. Timing requirement verification is done considering the entire chain of all applications involved in the propagation of the data. Feiertag et al. [3] presented a framework for the calculation of the end-to-end latency in multi-rate register-based systems. Different meanings of the end-to-end timing analysis depending on the functional behavior of sub-system units are proposed, where the notion of *maximum age of the data* (i.e control engineers) may be used, or as the first reaction to some data (the body electronics for instance). In our paper we are interested in the first meaning. Becker et al. [2] present methods to compute end-to-end delays based on different levels of system information by considering the job level dependencies and reachability between jobs. The maximum data age of the cause-effect chain is calculated considering a given data path.

Our contribution In addition to the results of [2] as well as of [3], the contribution of this paper resides in providing a policy allowing to all pairs of applications (producers and consumers) to exchange sufficiently fresh data. Our contribution is given for the case where FIFO-based circular buffers and different rates of the applications are considered.

The reminder of this paper is organized as follows. In Section 2 we present the system model and the associated notations. In Section 4 we present our contribution ensuring the data freshness maintenance. First numerical results are presented in Section 5. We conclude our paper in Section 6, where we provide also hints for future work.

In the reminder of the paper we use the word "task" to model an application and its temporal properties.

2. MODEL AND NOTATIONS

We consider S a set of n periodic tasks τ_i defined by (C_i, T_i), where $i \in \{1, 2, \cdots, n\}$, C_i is the worst-case execution time and T_i the minimal inter-arrival time of the task. All tasks have implicit deadlines, i.e., the deadline D_i of τ_i is equal to its period T_i. We denote by P the least common multiple of all periods and we have $P = lcm(T_1, T_2, ..., T_n)$, where T_i is the period of a task τ_i and n the number of tasks in the system. Each task τ_i executes $\frac{P}{T_i}$ number of instances $\tau_{i,j}$ within P, where j is the j^{th} instance of τ_i.

The tasks may belong to two different classes: producers and consumers. We may note that a task may be both producer and consumer, but for the sake of simplicity in the reminder of this paper we denote a task that is a producer by τ^p and a task that is a consumer by τ^c when we discuss the data the two tasks share.

The data dependence between tasks is modelled by precedence constraints between the tasks of S. A producer and a consumer are using a circular buffer B to exchange data and each data has its own buffer. The access policy in the buffer is considered to be FIFO and the buffer is characterized by its size l, which is the maximum number of data samples that the buffer may hold.

A data is characterized by a time stamp ts and time of issue ti. ts is the time when the data is registered within the buffer by its producer and it remains unmodified while the data is read by different consumers, and ti is the time a data is made available by a processing component at its output. It is then stacked into the corresponding memory buffer.

Figure 1 presents an example of circular buffer shared between $Component1$,$Component2$ and $Component3$ where the first is a writing component while the second and third are reading ones. Components interactions are managed into

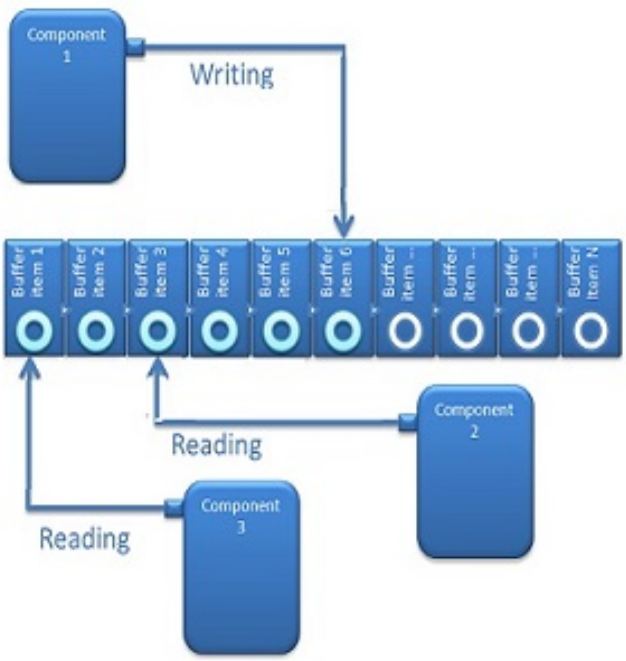

Figure 1: Example of a buffer as described in RTMaps software [1]

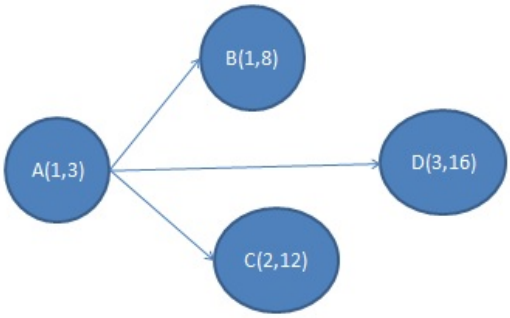

Figure 2: System model tasks

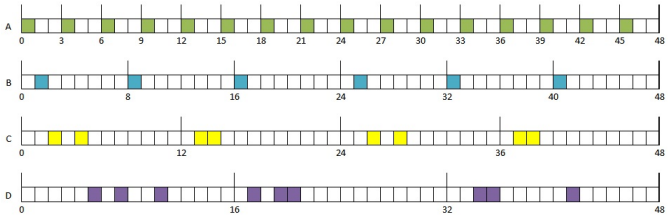

Figure 3: Scheduling results

a processing diagram. Each component has a processing loop which starts when the diagram is run. It then loops until the diagram execution is stopped. Within this processing loop, reading component reads data on its inputs, then processes the data, and finally writes the result into a memory buffer it is connected to.

A same data can be read by several consumers and a given consumer may read from different buffers depending on the number of inputs is has. The number of the memory buffers is equal output variables in the system since we allocate a memory buffer to each output data variable. The size of each memory buffer may deffer from one to another given that it is calculated taking into account the timing characteristics of tasks that utilize it.

A data is evicted from a buffer when the data is read by all its consumers or when it is overwritten by its producers due to the lack of space in the limited size buffer. The size of the buffer as well as the sampling rate have, then, a direct impact on the time a data can stay in the buffer before being read by **all** its consumers or before being overwritten by the producer.

3. DATA AGE WITHIN A FIFO BUFFER

We consider a set of tasks described in Figure 2, where A is a producer for the tasks B, C and D. Each task is characterized by a worst case execution time C and an inter-arrival time T. As stated in section 2, we consider a task system with implicit deadlines. We assume that the first instances of all tasks are simultaneously released at $t = 0$. In Figure 3 the scheduling results of the corresponding system model are presented. To schedule our task model we used SimSo, a simulation tool to evaluate Real-Time Multiprocessor Scheduling Algorithms. Here the execution of consumers instances are delayed by the execution of tasks with higher priority. At the same time a consumer task has to wait for the producer task in the case where the buffer is empty. If the size of the circular buffer is $l = 10$ and no subsampling rate is considered, then the data sample produced at time instant $t = 4$ is accessed by its slowest consumer, D, at time instant $t = 17$ even if at this same time 4 new samples are written into the memory buffer. In this situation, a delay of 13 units of time is obtained for this data in the buffer. In Figure 4 we present the end of writing time by the producer and the beginning of reading time by the consumer tasks equivalent to the proposed example. Reading the table by column allows us to observe the data instance used by the consumer and to observe the elapsed time between its writing and its reading moments. In the case that the consumer doesn't use the most recent written data, the following question arises: **Will the data still be useful for the functional correctness of the system?** After some time, due to the lack of buffer space, once the producer brings new samples, old data are overwritten whatever they are read by all consumers or not.

The case when data are overwritten before being read by all its consumers may not satisfy the entire critical mission of the vehicle, but it may prevent the system from a need of synchronizing data for instance. Thus, in some cases, this situation should be avoided. By comparison, the model used in[4, 5, 6] and others results calculating end-to-end timing analysis in the cause-effect chain are the particular case of our problem where the buffer is shared only between one producer and one consumer. Moreover, in our paper the necessary memory for data exchange management is allocated once at the beginning of the schedule and no dynamic memory allocation is performed during the execution of a task.

For each pair of producer and consumer $(\tau^p, \tau^c), \forall c \in (1, 2, ..., n_c)$, we denote τ^p as the producer task, τ^c as the consumer task and n_c the number of consumer tasks that read the data produced by τ^p. The producer τ^p is suscep-

Data samples produced by A										
Data sample number	1	2	3	4	5	6	7	8	9	10
Writing time by A	1	4	7	10	13	16	19	22	25	28
Reading time by B	1	8	16	25						
Reading time by C	2	13	26							
Reading time by D	5	17								

Figure 4: Data age calculation

tible to produce a data set D such that $(d_1, d_2.., d_\mu) \in D$ where μ is the capacity of the associated buffer. Each data sample d from D is characterized by two time stamps ts and ti such that $d_i^p = (ts_i, ti_i)$, where $i \in [1, \mu]$.

The focus in this paper is on cases where $T^p \leq T^C$. Here T^p stands for *sampling rate of the producer task* while T^c stands for *sampling rate of consumer task*. Scheduling priorities are assigned based to the sampling rates. The smaller is the period, the higher is the priority. Preemptions are allowed. Low priority tasks can be preempted and resumed later. When $T^p = T^C$, the high priority is given to τ^p to ensure data-dependency between task instances. We also assume the task worst-case response times to be known. In what follows, we propose a way to maintain the freshness of data being exchanged in the system while still preserving data synchronization. The latter requires not to overwrite data that are not yet read by all reader tasks.

4. DATA FRESHNESS MAINTENANCE

In this paper the focus is given to case where the writing task has a higher priority over readers. Thus, given that the tasks are periodic, then a scheduling interval is defined by a multiple of the LCM of all periods. Data reading happens at the activation of an instance of the reader task and gets the newest among available samples. In Section 5 we provide more details on an appropriate length for the schedulability interval.

4.1 Calculating the buffer size

For a schedulable system, all task instances are guaranteed to be released, to read input data, to process it and to write back results into memory before a deadline. As specified before, at each output data variable is associated a buffer from where reading components fetch data. Moreover, they may exist $(\tau_1^c, \tau_2^c, ..., \tau_\phi^c)$ that utilize data from B, where ϕ is the number of readers.

In[7] Kyoung-Soo We and al. propose *Most Recent Data Use* property to handle data from a buffer memory, where the memory buffer is overwritten by the most recently generated data. Even if, according to this property, the job that reads the memory buffer always uses the most recently generated data, it can not guarantee the synchronization of system data. This is an unwanted case for multi-rate systems since they exploit data from different sensors and individual tasks are often activated by independent clocks and often have different sampling rates.

In order to avoid this case, in this paper, we define the number of data samples to be maintained into the memory buffer and, for each reading task,we provide an algorithm which ensures the reading of the fresh data whenever an instance of the reader task is released (activated).

FIRST HYPER PERIOD

Time instant	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Executing task	A	B	C	A	C	D	A	D	B	A	D		A	C	C	A	B	D	A	D	D	A		
SSR_B	1	0	0	1	1	1	2	1	0	1	1		2	2	2	3	0	0	1	1	1	2		
SSR_C	1	1	1	2	1	1	2	2	2	3	3		4	4	0	1	1	1	2	2	2	3		
SSR_D	1	1	1	2	2	2	3	3	3	4	2		3	3	3	4	4	4	5	5	1	2		

Time instant	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Executing task	A	B	C	A	C		A		B	A	D	D	A	C	C	A	B	D	A			A		
SSR_B	3	0	0	1	1		2		0	1	1	1	2	2	2	3	0	0	1			2		
SSR_C	4	4	4	5	1		2		2	3	3	3	4	4	0	1	1	1	2			3		
SSR_D	3	3	3	4	4		5		5	6	6	6	7	7	7	8	8	2	3			4		

SECOND HYPER PERIOD

Time instant	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
Executing task	A	B	C	A	C	D	A	D	B	A	D		A	C	C	A	B	D	A	D	D	A		
SSR_B	3	0	0	1	1	1	2	2	0	1	1		2	2	2	3	0	0	1	1	1	2		
SSR_C	4	4	4	5	1	1	2	2	2	3	3		4	4	0	1	1	1	2	2	2	3		
SSR_D	5	5	5	6	6	6	7	7	7	8	2		3	3	3	4	4	4	5	5	1	2		

Time instant	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Executing task	A	B	C	A	C		A		B	A	D	D	A	C	C	A	B	D	A			A		
SSR_B	3	0	0	1	1		2		0	1	1	1	2	2	2	3	0	0	1			2		
SSR_C	4	4	4	5	1		2		2	3	3	3	4	4	0	1	1	1	2			3		
SSR_D	3	3	3	4	4		5		5	6	6	6	7	7	7	8	8	2	3			4		

Figure 5: Figure 6: Sub-sample rate computing results

From the meaning of the hyper period property, given the assumption that τ^p has a high priority compared to all τ^c, we can assume that the number of data produced by τ^p within P is constant. We consider this number to be the ideal capacity of the memory buffer and is given by $l = \frac{P}{T_{\tau^p}}$. However this is not a sufficient condition to guarantee the freshness of data being transmitted between connected components. In order to ensure the freshness of data being used in the system, we provide in the next section an algorithm to calculate a sub-sampling rate for each τ^c connected to a given memory buffer.

In the memory buffer data samples are accessed at different frequencies and each reading task must read the fresh data at the activation of each of its instances. Considering the fact that a component can have multiple inputs, this number of inputs is proportional to the number of memory buffers it is connected to. Data samples are continuously being inserted until the the buffer of concern gets full. Once the memory full, the new arrival data overwrites the oldest samples.

4.2 Calculating sub-sampling rate

For every pair (τ^c, B) we need to calculate $\lambda^{c,B}$ data samples that can be skipped every time an instance of τ^c is released. We call this *sub-sampling rate*. It can vary from a time window to another. Thus, it must be set online by considering the current state of the schedule. Within a task chain data exchange life cycle consists of 3 stages: *data request*, *data processing* and *data output*. Hence, we assume that once an instance of τ^c is released it fetches the data from the corresponding buffer, processes it and finishes its execution with a new data sample inserted into the buffer. We also assume that an instance of τ^c can only consume data produced before its activation. In additional to the parameters defined in the section 2, we define:

1. η^α the total of data samples produced
2. η^α_{act} the total of data samples produced at the activation of an instance of τ^c
3. η^B the number of data samples available in the buffer
4. $\lambda^{c,B}$ the sub-sampling rate of τ^c when reading from B
5. $l^{p,B}$ the size of the buffer between τ^p and τ^c

6. q^p and q^c the budget already consumed at the release of an instance of τ^p and τ^c respectively
7. τ^{prior} the task which priority is high at t time instant
8. ρ^c number of times an instance of τ^c was preempted during its execution

Algorithm 1 Calculate sub-sampling rate $\lambda^{c,B}$

Require: η^α,η^α_{act},q^p,q^c,τ^{prior} and ρ^c
Ensure: $\lambda^{c,B}$
Initialize: $\eta^\alpha \leftarrow 0$,$\eta^\alpha_{act} \leftarrow 0$,$q^p \leftarrow 0$,$q^c \leftarrow 0$,$\rho^c \leftarrow 0$
loop
  while $t<P$ **do**
    if $\tau^{prior} = \tau^p$ **then**
      $q^p \leftarrow q^p + 1$
      if $q^p = C^p$ **then**
        $\eta^\alpha \leftarrow \eta^\alpha + 1$
        $\lambda^{c,B} \leftarrow \lambda^{c,B} + 1$
        $q^p \leftarrow 0$
      end if
    end if
    if $\tau^{prior} = \tau^c$ **then**
      $q^c \leftarrow q^c + 1$
      if $\rho^c = 0$ **then**
        $\eta^\alpha_{act} \leftarrow \eta^\alpha$
      end if
      if $q^c = C^c$ **then**
        $\lambda^{c,B} \leftarrow (\eta^\alpha - \eta^\alpha_{act})$
        $q^c \leftarrow 0$
        $\rho^c \leftarrow 0$
      else
        $\rho^c \leftarrow \rho^c + 1$
      end if
    end if
    $t \leftarrow t + 1$
  end while
  return $\lambda^{c,B}$
end loop=0

5. FIRST NUMERICAL RESULTS

In Figure 5, **TI** stands for *time interval*, **SSR X** for sub-sampling rate of task X.

To illustrate our method, we consider a set of 4 tasks A,B,C and D, such that the task A is defined by $(1, 3)$, B by $(1, 8)$, C by $(2, 12)$ and D by $(3, 16)$, as described in Figure 2.

A is the producer and others tasks are consumers. Tasks are scheduled according to rate monotonic algorithm. Scheduling results are shown on Figure3. Simulations are performed within an interval of $10*P$, considering the size of the buffer to be l calculated according to equation **??**.

For a set $\Pi = (P^1, P^2, ..., P^k)$, where k is the number of the hyper periods. Lets $R^k_{slowest,1}$ be the worst case response time of the first instance of the slowest task within P^k and $R^k_{i,1}$ the worst case response time of the first instance of any of the task set (including the slowest). We showed that:

- The system behavior (in terms of data samples waiting to be read, by considering all tasks) repeats from a time interval bounded by $[R^k_{slowest,1}, R^{k+1}_{slowest,1}]$
- The system behavior (in terms of data samples waiting to be read by a given task) repeats from a time interval bounded by $[R^k_{i,1}, R^{k+1}_{i,1}]$

Data sample number	1	2	3	4	5	6	7	8	9	10
Writing time by A	1	4	7	10	13	16	19	22	25	28
Reading time by B	1	*	8	*	*	16	*	*	25	
Reading time by C	2	*	*	*	13	*	*	*	26	
Reading time by D	*	5	*	*	*	17				

Figure 6: Evaluation of our method

In the light purple part of Figure 5 we show the time interval indicating that the behavior of task B is identical in each hyper period, the one in sky blue for task C and finally the part in green for D.

In Figure 5 the number of data produced by A between two consecutive releases of tasks B,C and D is presented. So, at the release of any of the consumer tasks, data reading is performed on the data which is at the position equal to the sub-sampling rate compared to the recent reading.

For instance, the activation of the second instance of D happens at t=17 when there was already 4 unread data samples in the memory. It is preempted by A at t=18, resumes its execution at t=19 and finishes at t=21. At this moment there is a new unread data produced during the preemption time. That is why its sampling rate is now 1 instead of 0.

6. CONCLUSIONS AND FUTURE WORK

In this paper we present first results for calcuting the rate of sub-sampling of data based on their age. As future work, we would like to complete these first results by proving the existence of an appropriate schedulability interval, as well as studying the optimality of the sub-sampling calculation.

7. REFERENCES

[1] https://intempora.com/66-reader-types.html.

[2] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture - Embedded Systems Design*, 80:104–113, 2017.

[3] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A compositional frameworfor end-to-end path delay calculation of automotive systems under different path semantics. *CRTS*, 2008.

[4] T. Feld, A. Biondi, R. I. Davis, G. C. Buttazzo, and F. Slomka. A survey of schedulability analysis techniques for rate-dependent tasks. *Journal of Systems and Software*, 138:100–107, 2018.

[5] M. J. Friese, T. Ehlers, and D. Nowotka. Estimating latencies of task sequences in multi-core automotive ecus. *CoRR*, 2018.

[6] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Comput. Sci. Inf. Syst.*, 10(1):453–482, 2013.

[7] K. We, S. Kim, W. Lee, and C. Lee. Functionally and temporally correct simulation of cyber-systems for automotive systems. In *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*, pages 68–79, 2017.