Response time analysis with offset

Master Thesis in Information and Communication Technology

NGUYEN HONG VUONG

Department of Information & Communication Technology
UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
Intake 2011-2013

Supervisor: Professor Frank Singhoff
UNIVERSITY OF BREST

Tutor: Associated Professor Daniel Chillet
UNIVERSITY OF RENNES 1, ENSSAT

Contents

List of Figures

List of Tables

Abstract

Cheddar is a toolset based on real-time scheduling and queueing system theory, and is used to verify timing constraints of real-time systems. Currently the Cheddar team is looking for more distributed system support implemented into Cheddar.

CAN network has been used in automotive, and at least one generation of car is using offset for scheduling. However, no analysis tool was used, thus left the utilization of CAN network at a very low level. Because of that, the Cheddar team also want to implement offset support into Cheddar.

This work involves investigating, evaluating and implementing response time analysis algorithms with offset into the Cheddar toolset.

1 Introduction

The context of this research is the SMART project, which is a collaborative project between Lab-STICC UBO, Ellidis Technologies and Virtualys. Lab-STICC UBO, in the framework of SMART project, is developing the Cheddar toolset.

The Cheddar toolset is based on real-time scheduling and queueing system theory. The Cheddar toolset was developed to provide a tool for verifying timing constraints of real-time systems. The current implementation of Cheddar has very limited support for distributed system.

At the same time, in the framework of the SMART project, we need to model and verify a real-time system based on CAN network. The system is used in an automotive project.

Given the situation, we aim to investigate, evaluate and implement into Cheddar response time analysis algorithms that support CAN network in particular and distributed systems in general. The chosen technique is response time analysis (RTA) with offset.

The layout of this report: Section 2 gives background information on this work and works of other authors that relate to scheduling theory and CAN network. Section 3 shows the reason for choosing offset for this work. Section 4 gives the analysis on chosen algorithms. Section 5 gives the methodology for evaluation and the result of implemented algorithms. Finally, Section 6 gives hindsights and possible extensions to this work.

2 Related works

This work aim to investigate, evaluate and implement RTA algorithms with offset into Cheddar. This section gives brief view of the research background, many of which can be the target to extend this work.

2.1 Response time analysis with offset

The first foundation for RTA with offset was laid by Audsley et al.: both RM and DM priority assignment algorithms are not optimal for system with offset or abitrary deadline. Due to this, Audsley et al. developed an optimal priority assignment algorithm called ”Bottom up” [2].

Offset assignment is another pre-analysis problem alongside with priority assignment. In systems where offset are set by the scheduling algorithm (offset free systems), Grenier et al. developed several algorithms to assign offset and priority together - since offset free can make priority assignment non-optimal: the optimal method, dissimilar method or heuristics method [15].

Analysis algorithms for fixed priority systems with fixed offset were developed by Audsley et al. and Tindell [6] [7]. Later on, Palencia et al. developed a technique that allow dynamic and abitrary offset on fixed priority systems; after that Palencia et al. extended the technique to also support dynamic priority systems [9] [11].

Implementation-wise, Mäki-Turja et al. developed a technique to optimize the implementation of Tindell [7] and Palencia [9] algorithms for faster computation time. The increasement in performance was shown in benchmarks [12].

2.2 Response time analysis for CAN network

The foundation was laid by Tindell et al.: they first proposed a method to calculate worst-case response time of CAN messages by mapping CAN specified concepts and traditional RTA concepts [4]. Later, Dobrin et al. presented a method that transform RTA non-compatible offline scheduled transmission schemes into RTA compatible sets of messages [13]. Choquet-Geniet et al. on the other hand showed that systems with offset will also produce a cyclic schedule, thus is compatible with off-line scheduling [14].

Alongside with traditional priority-based RTA, CAN network nodes can also operate with a FIFO queue. There have been researches on RTA on CAN network that have some node are priority-based, where others are FIFO queue-based [17].

Very recently, the benefit of using offset to reduce WCRT has been proven using benchmarks by Grenier et al. and Yomsi et al. [16] [18].

3 Motivation

In this section, we will provide information on why Offset were chosen as the solution, and what the Cheddar team were looking for to implement into Cheddar.

3.1 Timing models

In an system using offset, related tasks are grouped into transactions that can arrive periodically or sporadically - thus transactions have an attribute called period, depicted as the minimal time between two successive arrival of the transaction.

Offset is defined as the time between the arrival of the transaction and the actual release of the task [6] [7].

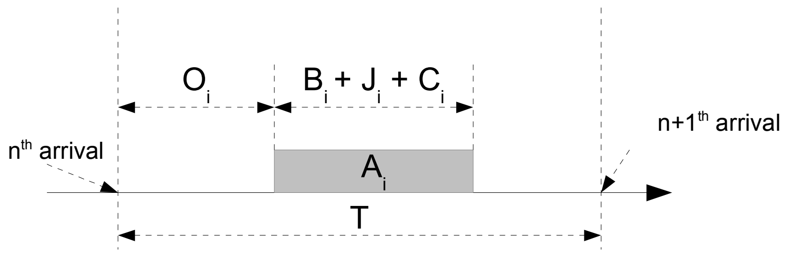

Figure 1: Timing model.

A CAN frame is modelled as a task with the following attributes [3] [18] (Fig. 1):

- *Period* T: frames are sent by application tasks running in CAN nodes. Thus, frames inherit the "period" attribute of such tasks.
- *Jitter* J: release jitter is another characteristic of application tasks that frames inherit from.
- *Capacity* C: the time needed to transmit a frame end-to-end. It is deliverable from the size of the frames (including CAN-specified additional data) and the network bandwidth.
- *Deadline* D: a relative deadline for a frame to be transmited.
- *Blocking time* B.
- *Worst-case response time* R: the longest time taken to deliver a frame end-to-end.

3.2 Motivation of using offset

The classic response time analysis techinques (RMA, DMA) assume that the tasks are independent. In fact, the dependencies between tasks can still be expressed via their priorities. Depends on the technique used, the priority of each task can be easily changed by changing the appropriate attribute of the task. However, this can greatly affect the design of the system being scheduled itself.

Example 3.2.1:

- Fixed priority system
- Task A preceed task B

In a fixed priority system, the optimal priority assignment is rate monotonic. Thus, the solution is to assign task A with a shorter period than task B (Fig. 2), but the business logic behind the two tasks might prevent such design.

The problem can also be solved using only offset: we can set the offset of task A greater than task B.

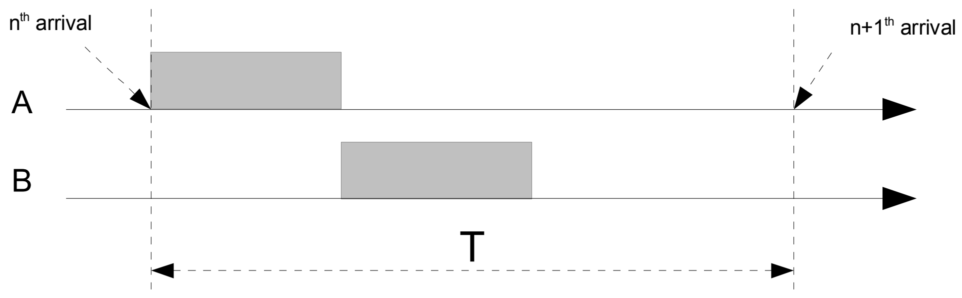

Figure 2: A has higher priority than B, thus preceed B.

In this example, offset works like a replacement for priority - however that is not all about offset. Let us extend the constraints in the previous example:

Example 3.2.2:

- Fixed priority system
- Task A preceed task B
- Task A and task B mutually exclude each other

In the ideal case, mutual exclusion are automatically taken care of once the priorities of tasks are set. In other words, in non-preemtive systems or systems designed that no preemption would occur (higher priority tasks would always be released before lower priority tasks), designer can forget about mutual exclusion. In reality, even in the case that the two tasks are of the same priority, no system would execute a preemption since there will be overheads (context switch or cache for example).

Back to the example at hand, the precedence constraint is taken care of in the same way we deal with example 3.2.1. The priority approach will have problems with the mutual exclusion constraint: the designer will have to design the system in a way that prevent successive arrival of task A (higher priority) preempt an executing task B (lower priority) (Fig. 3). Clearly, using only priority is not the ideal solution to meet the constraints in this example.

The offset approach, however, can take care of mutual exclusion by using previously unused priority attribute of the tasks. The solution then become (Fig. 4):

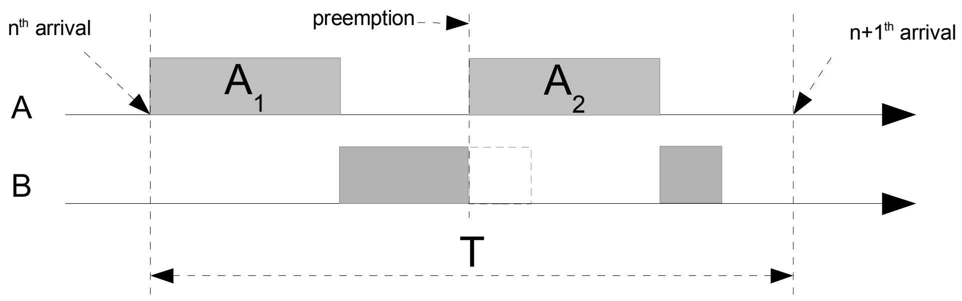

Figure 3: A has higher priority than B, thus preempt B.

- Give A a smaller offset than B so that A preceed B.
- Give B an offset so that even in the case that A experiences WCRT, A still finishs before B is released. ($O_B \geq O_A + R_A$)
- Give B higher priority than A so that A can never preempt B.

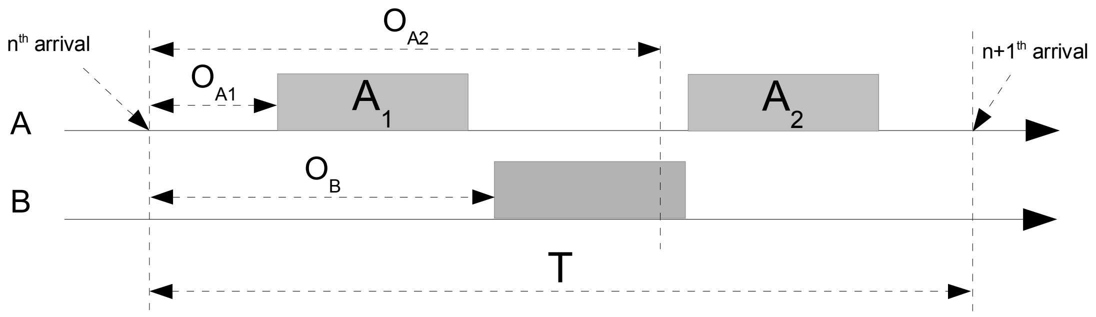

Figure 4: A has lower priority than B, thus cannot preempt B.

In reality, there can be a mandatory delay between the finish of task A and the begin of task B. Such delay can be introduced due to various reasons: task A and B might be exchanging messages, and there must be a certain amount of time to deliver the message. Another common use case is the I/O time: since the task cannot suspend itself, the designer can make a decision of breaking the task into two. Often enough the reason is the I/O time, which can be large than the execution of the task itself. The idea is to release the CPU during I/O, since the CPU will be free anyway. Thus the example can be extended as follow:

Example 3.2.3

- Fixed priority system
- Task A preceed task B
- Task A and task B mutually exclude each other
- There must be a minimum of Δ time units after A is finished and before B is released

With priority alone, such timing pattern between A and B is unachievable. The offset approach (Fig. 5), however, is sufficient with a minimal change to the solution from example 3.1.2:

$$O_B \geq O_A + R_A + \Delta_{AB}$$

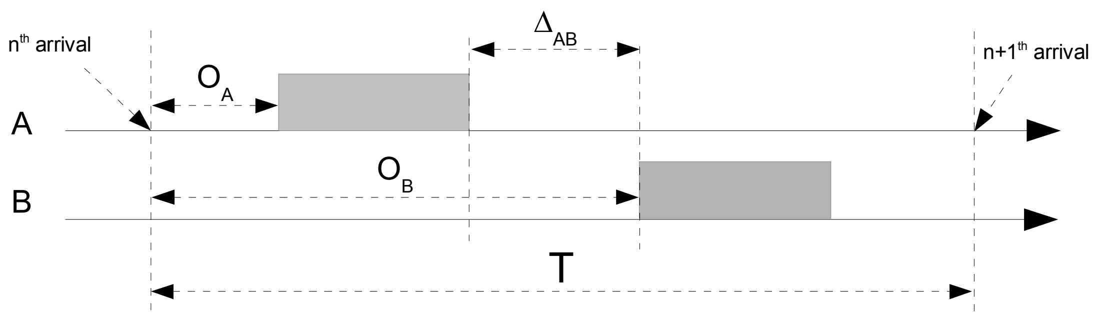

Figure 5: Using offset to express mandatory delay between two tasks.

In addition to task dependencies, offset can also be used to support systems with asynchronous communication between tasks.

Communication between task on a same system can be achieved using the producer-consumer model. Such design would require a priority ceiling protocol to control accesses to shared resource - unless there are no concurent accesses to the shared resource. Such condition can easily be achieved using offset: let us say that task A and task B access to the same shared resource. We can assign task B an offset that is greater than the worst case response time of task A plus task A offset. This way task A and task B will never be executed in parallel, thus the use of a PCP is not needed. This is actually the same solution as of example 3.2.2 (Fig. 4).

Tindell [8] mentioned an use case for offset in interprocessor communication. The system use a client-server model: client tasks send messages to a server task, awaiting for the reply. Server tasks wait for a message arrival and activate itself, execute the business logic and return the result to the client task.

To implement such architecture, a solution is to transform client tasks by breaking each of them into two child tasks: one that sends, and one that receives. The dependencies between such two task is exactly the same as in example 3.2.3 (Fig. 5); in other words offset can be used.

The previous design has a flaw however: each server task can only receive one message - in other words, one server task for each message sent. This introduce response time and resources overhead to the system, due to the fact that multiple instances of code are loaded into the RAM, which takes time and trashes resources. The design can be further improved by using the same technique on the server tasks: split server tasks into two parts. Part one are tasks which are spawned everytime a message arrive, and it will write the message into a buffer (a shared protected object). The second part is a single task that executes business logic and sends reply. We set an offset to the second part, during this offset time the buffer is filled with messages to be processed (Fig. 6).

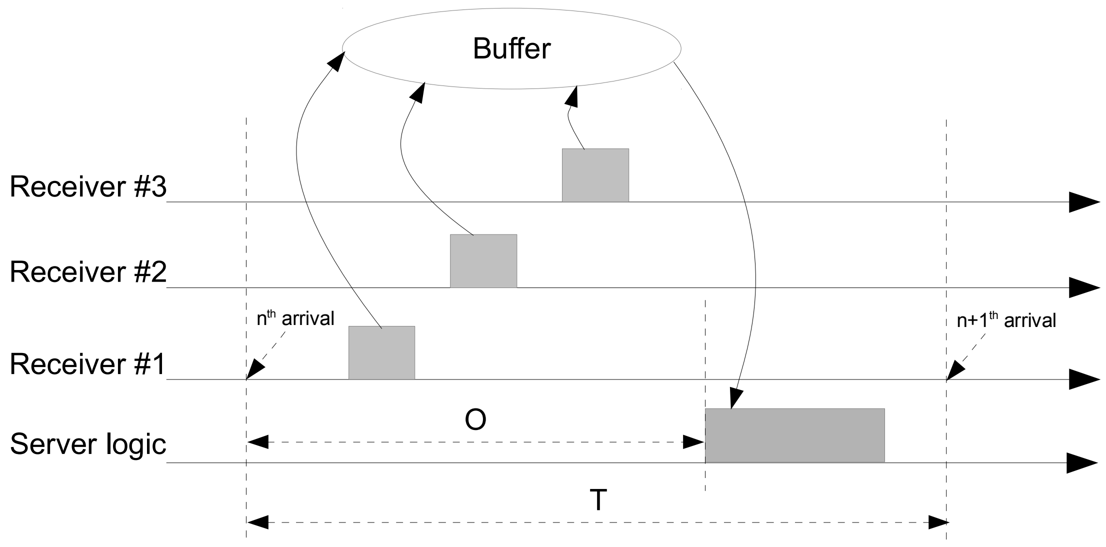

Figure 6: Server architecture: multiple sporadic request receivers, one result sender

3.3 Motivation of using offset on CAN network

3.3.1 CAN network message RTA

We know that response time analyses are indeed applicable on CAN network messages [3]; and the mapping of CAN network model to a system compatible with response time analyses is also simple. In addition the model can also be extended to add support for offset [18].

According to Yomsi et al., A CAN network message is modelled as a tuple (J, C, O, D, T, P) [18]

- J: the maximum jitter
- C: the capacity - the worst-case end-to-end transmission time.
- O: the offset
- D: the relative deadline
- T: the period of the transmission
- P: the priority of the message

3.3.2 Solving CAN network practical problems with offset

Grenier et al. [16] pointed out some characteristics of CAN networks. Without offset, the WCRT of a message increase drastically with the load. At the same time, although offset has been used by car manufacturers and analysis tools that support RTA with offset has been around for a long time, these analyses tools were not used in practical (this is also one of the problem that Cheddar is trying to solve). Because no analysis tools were used, to guarantee periodic/sporadic messages' deadline are met, the CAN bus utilization is typically keep at a low level (30-40 percents) leaving room for aperiodic messages.

In theory, a task will suffer WCRT when it is released simultaneously with all tasks of higher priority than itself. On the other hand, the use of offset can prevent such situation, and therefore, in theory, reducing the WCRT of a task. Consequently, the theoritical maximum bus utilization can be more optimistic. To prove this theory, Grenier et al. [16] and Yomsi et al. [18] did a number of benchmarkings. Their experiments showed the expected theoritical benefit of using offset. Grenier et al. concluded based on the experiments that with offset, bus utilization can be raised from 30% to 60% without affecting the performance of the CAN network systems. The experiments of Yomsi et al., on the other hand, showed a reduction in WCRT of tasks after introducing offset. While the higher priority tasks gain less from offset, the lower priority tasks can have a gain of 59-65% in WCRT.

3.4 Conclusion

Previous works have shown that response time analysis is possible on CAN network systems, with the mapping fairly simple; at the same time offset, being able to solve practical problems of CAN network systems, is indeed used for CAN network systems in automotive. However, analysis technique weren't used due to the lack of reliable tool (The available option Netcarbench implements undisclosed algorithms).

On the other hand, offset is clearly applicable on distributed system - while we also want to extend Cheddar's currently limited support for RTA on distributed systems.

4 Response time analysis with offset

In this section, we explain about the 3 algorithms that were implemented into Cheddar and give some comparisons. The chosen algorithms were the works of Audsley et al. [6], Tindell [7] and Palencia et al. [9].

4.1 Computational models

The computational models used in the 3 algorithms were quite similar to each other, with only differences in constraints on some attributes.

An additional model was added to Cheddar: *transaction*, which is a group of tasks. Currently, in Cheddar, there are two types of task group; one of them is transaction. Transaction can arrive either periodically or sporadically, thus is characterized by its ”period” (denoted T which is the minimum time between two successive sporadical arrivals).

4.1.1 Similarities

The 3 algorithms agreed on the transaction model.

The *task* model, in the 3 algorithms, retains the old attributes from the periodic task model (capacity C, deadline D, blocking time B, release jitter J) *except* period. The reason is that the period attribute became redundance after the introduction of the *transaction*. To support offset, a new attribute ”offset” denoted O is added to the model.

4.1.2 Differences

There are different constraints on the task model, due to the newer algorithms try to generalize the older one. In Audsley and Tindell algorithms, only the deadline is abitrary. In Palencia algorithm however, the task model can have abitrary deadline, jitter and offset. (Abitrary deadline means the deadline of a task can be greater that its period. In other words, multiple instances of a task can be active in the system at a given time. Palencia called such instances ”jobs”, while the other two didn’t mention about them.)

To support task transformation, Tindell task model have an ”every” attribute denoted ”e” with the meaning: the task will arrive (at most) once every ”e” arival of the transaction it belong to.

4.1.3 Grouping tasks into transactions

Task grouping is a pre-analysis problem that system designers have to solve. How tasks are grouped will affect the result of the analysis; an example is given in section 5 to demonstrate this aspect.

The principle is that tasks that have relationships with each other are grouped into a transaction. On the other hand, since transactions have ”period”, every tasks that share the same period can be grouped into a transaction. Additionally, while tasks that have relationship with each other often have the same period, there are cases where the period of a task is a multiple of the period of the related task. In this case, task transformations can be performed in order to obtain task sets with the desired periods. Possible ways to transform a task:

- Breaking a task into multiple tasks. The period is then multiplied by the number of child tasks. The attributes of child tasks are then taken care of as in example 3.2.2 or 3.2.3 (please refer to section 3).
  - Ex.: break task A of period x into task A1 and A2 of period 2x
- Cloning a task. The period is then multiplied by the number of clones. Offsets can be used to simulate the periodic timing pattern between such clones.
  - Ex.: clone task A of period x into 4 instances of period 4x (Fig. 7)

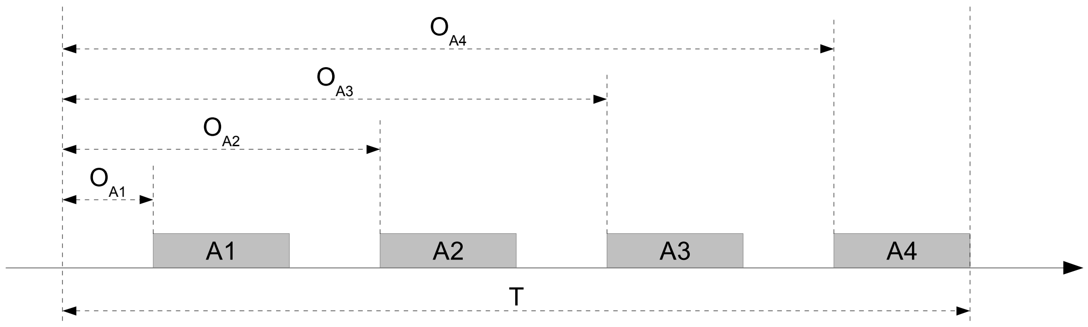

Figure 7: Using offset to simulate periodic timing pattern between clones.

- Using the ”every” attribute (in Tindell’s algorithm)
  - Ex.: transform task A of period 4x into A’ of period x and every 4

4.2 Level-i busy period and critical instant

There are two important concepts that will be used in later sections: the level-i busy period and the critical instant. (Fig. 8)

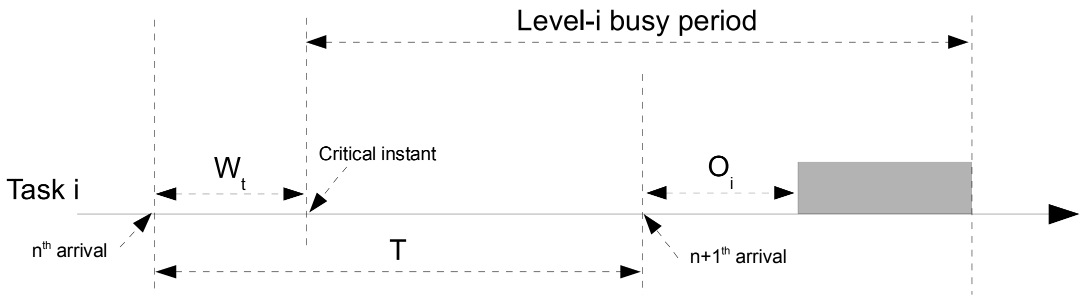

Figure 8: Level-i busy period and Critical instant.

- w_i: the width of the level-i busy period

- W_t: the phasing between the critical instant and the lastest release of t before the critical instant
- t: the transaction task i belong to
- T: the period of t (and task i)
- O_i: the offset of task i

4.2.1 Level-i busy period

Level-i busy period is defined as a continuous time interval $[t1, t2)$ where tasks of lower priority than i cannot be activated. In other words, a level-i busy period consist of instances of task i and all the task of higher priority than i. The width of the level-i busy period is an important factor that will be used in the algorithm to calculate the corresponding response time of task i.

4.2.2 Critical instant

All response time analysis algorithms start evaluating at the critical instant. A *critical instant* is the start of a level-i busy period. Note that the use of offset make it impossible to have a simultaneous release of task i and all the higher priority tasks: tasks of the same transaction that have different offset may never be released simultaneously. Because of this, a transaction with n different tasks may have n variations where the release of one of the n task fall into a critical instant (Fig. 9). Thus, multiple critical instants (corresponding to multiple level-i busy periods) exist for a given task i, and response time analysis algorithms for systems with offset begin with finding the right critical instant that leads to the worst case response time.

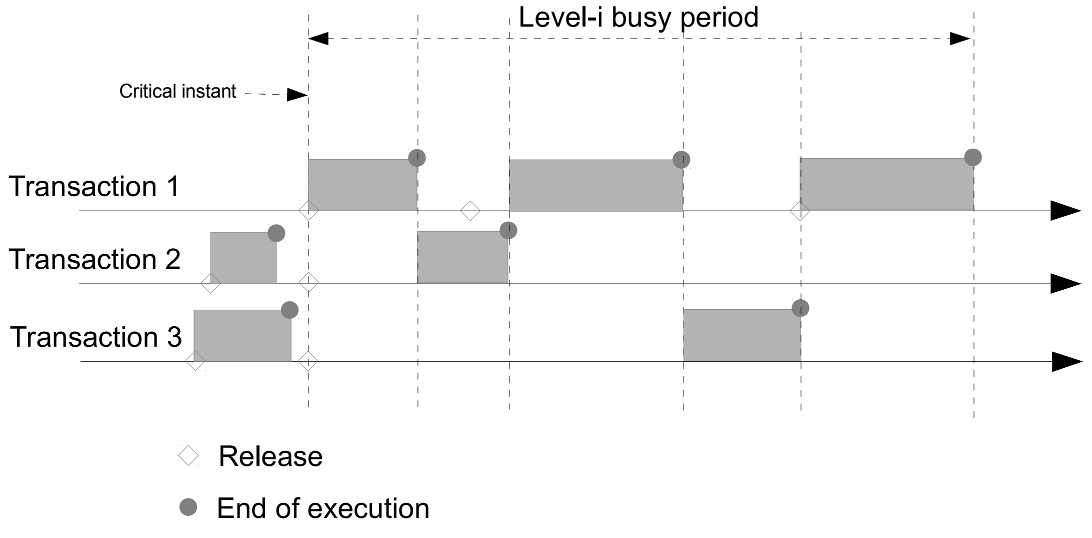

Figure 9: A critical instant / level-i busy period variation

4.3 Priority assignment

Priority assignment according to rate monotonic or deadline monotonic are found non-optimal for systems with arbitrary deadline tasks and systems with offset [6]. Audsley derivated an optimal priority assignment algorithm called ”bottom up” [2]; the algorithm is used in all 3 RTA algorithms mention in section 4.

4.4 The algorithms

The algorithms of Audsley et al. [6], Tindell [7] and Palencia et al. [9] share common points in the approach. There are differences in details however, since the algorithms have different assumptions and constraints. In this section, we will first present the general approach of the 3 algorithms. After that, Tindell's algorithm will be used to demonstrate the approach.

4.4.1 The general approach

Calculate WCRT of a task base on level-i busy periods

The width of the level-i busy period plays an important role in finding the respective task i response time. The equation used in the 3 algorithms were something similar to the below:

$r_i = w_i + W_t - aT - O_i$ (1)

- a: the number of releases of t during the level-i busy period
- r_i: the response time of task i

For example, look at Figure 8.

A variation is characterized by a task j which is released at a critical instant. Thus, we can deduct that said task is released right after the phasing W_t (see Figure 8). In other words, $W_t = O_j + J_j$.

Note that in the case of sporadic transactions, WCRT will only happen when such transactions are released at the fastest pace possible (in other words, sporadic transactions are released without the jitters of the events that trigger them) [7].

The worst-case response time R_i is then the maximal value of r_i corresponding to the level-i busy periods. The problem now narrows down to determining which level-i busy periods to examine and how to calculate the width of them.

Which level-i busy periods to evaluate?

We know that in system that make use of offset, multiple critical instants exists. As per the definition, a critical instant is the start of a level-i busy period, so multiple level-i busy periods exists.

Tindell [7] and Palencia et al. [9] proposed ”exact” methods of computing worst-case response time. The principle is to examine every possible level-i busy periods - or in other words, to examine every possible combinations of contribution of every transactions. This guarantee to find the exact worst-case response time, but is computational infeasible for non-trivial problems: suppose we have a system with m transactions. Transaction t_1 contains n_1 tasks, transaction t_2 contains n_2 tasks and so on. The total number of possible combinations that have to be evaluated is then

$n_1 n_2 n_3 ... n_m$.

Note that all 3 algorithms allow abitrary deadlines, in other words multiple instances of a task might present in the level-i busy period. Also note that the previous instances of task i will also contribute interferences to the level-i busy period. These instances are treated separately: Audsley et al. and Tindell evaluate the number of said instances increasingly from 0; while Palencia et al. developed a technique to compute the exact number of instances. For readability, we truncate the treatment for such instances from the equations below.

Because the total number of level-i busy periods can be too large to examine each and every of them, "tractable" methods was developed by Audsley et al., Tindell, and Palencia et al. The principle is simple: we assume that a transaction have maximal contribution to the level-i busy period that lead to WCRT. Of course this will produce more pessimistic results than the exact method, however the number of level-i busy periods to be examined is reduced tremendously. In addition, according to Tindell [7] his experiment showed that tractable analysis delivered the same result as exact analysis 93% of the time.

To reduce pessimism of tractable analysis, some transactions can be evaluated in the exact manner, and the rest in tractable manner. The 3 algorithms chose only the transaction t of task i to apply exact analysis.

Calculate the width of a level-i busy period

A level-i busy period consist of contributions of transactions in the system. The width of a level-i busy period is then the sum of contributions of the transactions.

$w_i = \sum_{\forall t \in trans} I_t$ (2)

- trans: set of transactions in the system
- I_t: (I)nterference of t to the execution of task i (or in other words contribution of t to w_i).

Thus, the problem narrow down to calculating the contribution I_t of a particular transaction t.

We know that a transaction can have more than one variation of contribution. In tractable analysis, the "contribution of a transaction" is the maximal value of all variations:

$I_t = max_{\forall j \in tasks(t) \cap hp(i)}(I_{t,j})$ (3)

- tasks(t): the set of tasks that belong to transaction t
- hp(i): the set of tasks that have higher priority than i.
- $I_{t,j}$: the interference of t with the task $j \in tasks(t)$ characterize the variation.

Because of that the problem is narrowed down to determine the contribution of a variation. On the other hand, transactions are collection of tasks; the contribution of such tasks make up the contribution of a transaction. Thus, the problem is to find the contribution of tasks in a variation.

The contribution of a task j can be calculated simply by multiplying the capacity of said task C_j and its number of releases k_j during the level-i busy period: $k_j C_j$. The number of time a task is released during level-i busy period will be the highest when it is first released at the critical instant, in other word:

$k_{max} = \left\lceil \frac{W+w_i-O_j}{T} \right\rceil$ (4)

when $O_j + J_j = W_t = O_i + J_i$.

W_t will be used in a ”normalize” function to calculate k from k_{max} otherwise.

Now we have the equation to calculate the contribution of a transaction:

$I_t = \sum_{\forall j \in tasks(t) \cap hp(i)} k_j C_j$ (5)

Note that w_i is a function of I_t (equation 2) and I_t is also a function of w_i (equation 3, 4, 5). We use the iterative method to calculate the approximation of w_i until we have two identical successive values, or until we find that the tasks are unschedulable.

4.4.2 Tindell's algorithm

This section demonstrate the approach of Tindell.

- $w_{i,q}$: the width of the level-i busy period with q extra instances of task i
- ti: the transaction that task i belong to
- e_i: the attribute ”every” of task i
- $I_{i,t}$: the total contribution of transaction t to $w_{i,q}$
- $K_{j,t}$: the number of instances of task j of transaction t during $w_{i,q}$
- v_j ; $v_{i,ti}$: the ”normalize” function to calculate $K_{j,t}$ from k_{max}

Tindell's algorithm to calculate r_i starts by selecting the number of extra instances of task i and the task j (to calculate exact analysis on transaction t):

$$r_i = \max_{q=0,1,2,3,\cdots} \left(\max_{\forall j \in tasks(ti)} \left(w_{i,q} + O_j + J_j - T_{ti}\left(qe_i + v_{i,ti}\right) - O_i \right) \right)$$ (6)

Notice that $qe_i + v_{i,ti}$ is the equivalent of a from equation (1); and $O_j + J_j = W_t$. To solve equation (6), we need to find $w_{i,q}$:

$$w_{i,q} = B_i + (q+1)C_i + \sum_{\forall t \in trans} I_{i,t}$$ (7)

Equation (7) is Tindell algorithm's equivalent of equation (2), with the additional treatment for instances of task i. The next problem is to calculate $I_{i,t}$:

$$K_{j,t} = \left\lceil \frac{W_t + w - O_j - v_j T_t}{e_j T_t} \right\rceil \quad (8)$$

$$I_{i,t} = \sum_{\forall j \in hp(i) \cap tasks(t)} K_{j,t} C_j \quad (9)$$

Equation (8) and (9) is Tindell algorithm's equivalent of equation (4) and (5) respectively. $w_{i,q}$ thus can be obtained by calculating a series of approximation, until there are two identical successive values, or until we find that the tasks are unschedulable.

4.5 Conclusion

In this section we have seen the general approach to RTA with offset of Audsley et al., Tindell and Palencia et al. Due to this, we can predict that the performance of the 3 algorithms would be quite similar to each other - the differences would be from how they handle small details, since they have different set constraints.

5 Implementation and evaluation

In this section, we explain the model design to support the implemenation of the algorithms mentioned in section 4. We then give the evaluation strategy and the result of evaluating said algorithms and implementations.

5.1 The design of Cheddar toolset

The architecture of Cheddar framework consist of two layers: the low-level layer which is dedicated to data management, and the high-level layer which is Cheddar domain specific. In the section below, the design decision will have to be implemented in the high-level layer [1].

Cheddar is implemented in a model-driven manner: significant parts of the components in high-level layer are generated from the Cheddar internal models called EXPRESS models. The models are written in EXPRESS language.

5.2 Design

To support response time analysis with offset, the EXPRESS models had to be changed:

- Task model: add an offset table and an "every" attribute. Offset table must be used instead of a single value to support dynamic offset which can vary in a range.
- Transaction model: a type of Task Group along side with Multiframe (which is outside of the scope of this work).

Figure 10 shows the new EXPRESS models. Offset tables and "every" were added to existing models, while two new models were added: Generic_Task_Group and Transaction_Task_Group.

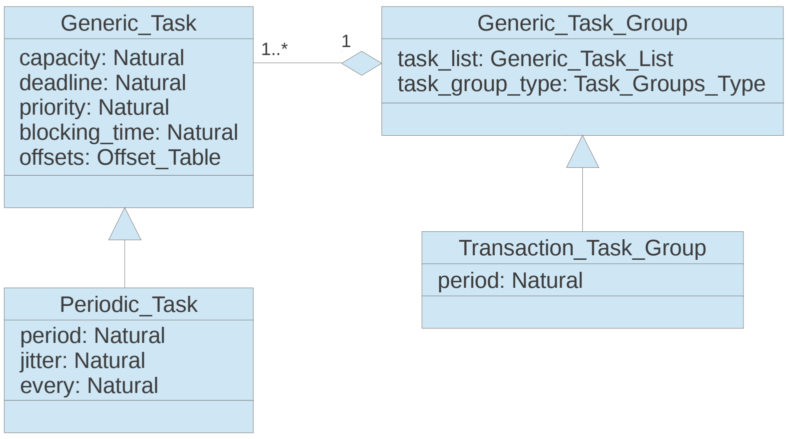

Figure 10: New Cheddar EXPRESS models

5.3 Implementation

The implementation is written as part of the Cheddar framework, and is currently available at the offset branch of Cheddar:

`http://beru.univ-brest.fr/svn/CHEDDAR/branches/vuong-offset/`

The implementation of the 3 response time analysis algorithms is available in a single Ada package: **Feasibility_Test.Worst_Case_Response_Time**. At the time of writing, only Audsley and Tindell algorithms are working correctly. Palencia algorithm still requires further work to make it runs and also support for dynamic offsets are not completed.

5.4 Evaluation

5.4.1 Evaluation strategy

Because the implementation of Palencia's algorithm isn't completed, we will only evaluate Audsley and Tindell's ones. There are two main criterias to evaluate the algorithms and the implementations: the precision and the performance.

The precision of the algorithms and the implementations are evaluated using the example task sets that are taken from published papers. Although it would be best to evaluate them with a real usecase of CAN network, we couldn't get it done within the time limit.

The performance of the implementations are evaluated by benchmarking against large task sets of increasing size.

5.4.2 Evaluating the precision

Example task set #1: Xu and Parnas'

This task set is presented by Xu and Parnas and is claimed that it is unschedulable using either fixed or dynamic priority assignment [5].

Task	C	D	O	Pri	r	r_{old}
A	30	110	51	5	110	**150**
B	30	40	11	1	30	30
C	30	30	60	3	30	**70**
D	10	59	41	2	10	40
E	50	50	90	4	50	**120**

Table 1: A task set that is unschedulable without offset

The system consists of 5 tasks and 1 transaction. The detail of the tasks are given in the table 1 above. The transaction has a period of 161 units of time.

- C: capacity
- D: relative deadline. **Bold** means the deadline is missed
- O: offset
- Pri: priority (1 is highest)
- r: worst-case response time

- r_{old}: response time without the use of offset using fixed priority

This example demonstrates the benefit of using offset: without offset, most of the tasks would miss their deadline - while with offset, all of them didn't. Task E has its response time reduced by more than half and fits nicely in its deadline. Notice how lower priority tasks benefit more from offset than higher priority task.

In this example, both Audsley and Tindell algorithms delivered the same results as in the source paper.

Example task set #2: Audsley's without task transformation

Task	C	D	O	pri	T	$r_{Audsley}$	$r_{Tindell}$
A	1	1	4	1	10	1	1
B	1	2	5	4	10	**15**	**15**
C	5	6	0	2	20	**7**	6
D	8	9	7	3	40	**17**	**15**
E	8	14	27	6	40	**40**	**40**
F	6	30	0	5	40	30	30

Table 2: A poorly designed system leads to pessimistic response time

This example is taken from Audsley's paper about optimal priority assignment [2]. The system consists of 6 tasks, belonging to 3 different transactions. Task A and B belong to a transaction with T = 10; task C belongs to a transaction with T = 20; the rest belong to a transaction with T = 40.

This example demonstrates how pessimistic the analysis result can be for a poorly designed system. Audsley's algorithm delivered a more pessimistic result than Tindell's one in this example: with 4 tasks missed their deadlines - comparing to 3 tasks of Tindell.

Example task set #3: Audsley's with task transformation

Now consider a system that is designed with task transformation. All the task now are grouped into 1 single transaction with T = 40. The method of task transformation is to clone tasks and give them the right offsets to simulate their periodic behaviour.

Task	O	D	r
A	4	1	1
	14	1	1
	24	1	1
	34	1	1
B	5	2	2
	15	2	2
	25	2	2
	35	2	1
C	0	6	6
	20	6	6
D	7	9	9
E	27	14	13
F	0	30	30

Table 3: Task transformation makes response time of tasks more optimistic

In this example, both Audsley and Tindell's algorithms delivered the same results, matching the hand-calculated result in Tindell's paper [7]. This time all the tasks met their deadlines respectively. Notice the huge gain in response time of task E, from 40 reduced to 13 units of time.

5.4.3 Evaluating the performance

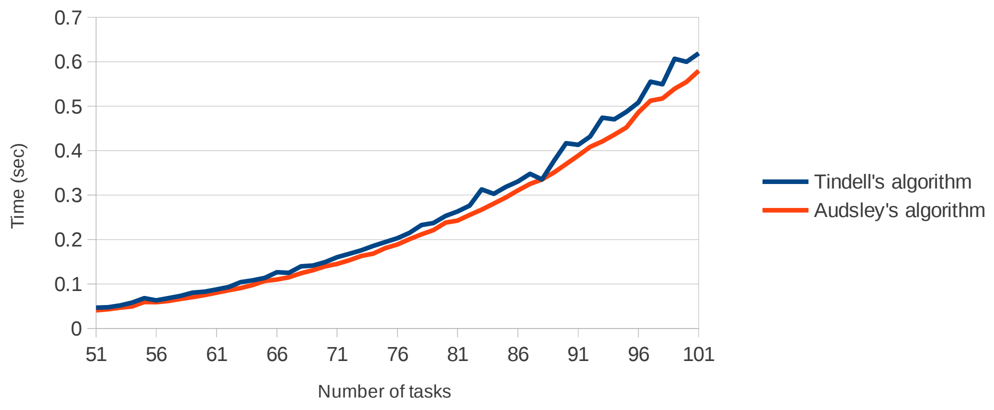

Figure 11: A performance benchmark using randomized task sets of increasing size

The evaluation process was as below:

1. Generate a random task set of desired size
2. Process the task set 10 times with each algorithm's implementation
    - Measure the time taken each time
    - Take the average time
3. Increase the size of the task set and go back to 1.

To avoid benchmarking pitfalls, multiple measures were used: A sample task set is evaluated multiple of times to eliminate the presence of any external influences. Also the evaluation was not done with the Cheddar simulation tool but with a simple unit test that make use of the Cheddar core framework. The reason for doing so is to reduce overhead as much as possible: only the execution time of the Ada procedure that implement the algorithms are measured.

Because the algorithms were following a general approach to the problem, the curves were of the same shape (increasing exponentially). Audsley's algorithm however showed that it is faster than Tindell's one most of the time, while can be more pessimistic (refer to section 5.2.2).

5.5 Conclusion

There are two things that make a good software: its design and its implementation. In this section we have seen that Audsley and Tindell algorithms are, by giving the same result as in their algorithm's respective publications, working as expected. Audsley algorithm is known to produce a more pessimistic result than Tindell algorithm in some cases, however it is not clear if that is the expected result. The benchmark however, showed that Audsley algorithm is always faster than Tindell algorithm.

Further investigation might be required, as well as an evaluation against a real usecase of CAN network.

6 Conclusion and future work

We have implemented response time analyis algorithms with offset into Cheddar, and have verified their precision and performance. In hindsight, in this work we didn't really tackle distributed system nor CAN network - we have implemented algorithms that are compatible with them instead: tasks having offset relationship with each other might of course be distributed on different processors; scheduling with offset is indeed tranformable into offline scheduling that stay in CAN network nodes. This work has laid the foundation for future works in Cheddar.

We have seen that offset can be used to increase schedulability, by negating the traditional critical instant. We also have seen what can be done with offset to model differents real-life constraints in section 3. The application of offset is vast; offset support in Cheddar allow users to model more complex real-life systems.

As mentioned in section 2, there can be many possible future works for this research. First and foremost however, is to complete Palencia algorithm and add support for dynamic offset. After that, we will address pre-analysis system design problems such as providing tools to assist designer in assigning offsets and priorities - as this would be in demand. Last but not least, we would like to seriously tackle the problems of CAN network schedulability analysis.

References

[1] F. Singhoff, A. Plantec, P. Dissaux, J. Legrand, *Investigating the usability of real-time scheduling theory with the Cheddar project*, Journal of Real Time Systems, volume 43, number 3, pages 259-295. November 2009. Springer Verlag. ISSN:0922-6443.

[2] N. C. Audsley, *Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times*, YCS 164, Dept. Computer Science, University of York, 1991.

[3] K. Tindell, H. Hansson, and A. Wellings, *Analyzing Real-Time Communications: Controller Area Network (CAN)*, Real-Time Systems Symposium, 1994., Proceedings.. IEEE, 1994.

[4] K. Tindell, A. Burns, *Guaranteeing message latencies on Control Area Network (CAN)*, Proceedings of the 1st International CAN Conference. 1994.

[5] J. Xu, D. L. Parnas, *On Satisfying Timing Constraints in Hard Real-Time Systems*, Software Engineering, IEEE Transactions on 19.1 (1993): pages 70-84.

[6] N. Audsley, K. Tindell, A. Burns, *The end of the line for static cyclic scheduling?*, Proceedings of the 5th Euromicro Workshop on Real-time Systems. 1993.

[7] K. Tindell, *Adding Time-Offset to Schedulability Analysis* Real-Time Systems Research Group, Department of Computer Science University of York, England, 1994.

[8] K. Tindell, J.Clark, *Holistic schedulability analysis for distributed hard real-time system*, Microprocessing and microprogramming 40.2 (1994): pages 117-134.

[9] J.C. Palencia, M. González Harbour, *Schedulability Analysis for Tasks with Static and Dynamic Offsets*, Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE. IEEE, 1998.

[10] J. Goossens, *Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints*, Diss. PhD thesis, Université Libre de Bruxelles, 1999.

[11] J.C. Palencia, M. González Harbour, *Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF*, Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on. IEEE, 2003.

[12] J. Mäki-Turja, M. Nolin, *Faster Response Time Analysis of Tasks With Offsets*, Proc. 10th IEEE Real-Time Technology and Applications Symposium (RTAS). 2004.

[13] R. Dobrin, G. Fohler, *Implementing Off-line Message Scheduling on Controller Area Network (CAN)*, Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on. IEEE, 2001.

[14] A. Choquet-Geniet, E. Grolleau *Minimal schedulability interval for real-time systems of periodic tasks with offsets*, Theoretical computer science 310.1 (2004): pages 117-134.

[15] M. Grenier, J. Goossens , N. Navet, *Near-Optimal Fixed Priority Preemptive Scheduling of Offset Free Systems*, 14th International Conference on Real-Time and Networks Systems (RTNS'06). 2006.

[16] M. Grenier, L. Havet, N. Navet, *Pushing the limits of CAN - scheduling frames with offsets provides a major performance boost*, 4th European Congress on Embedded Real Time Software (ERTS 2008). 2008.

[17] R. I. Davis, S. Kollmann, V. Pollex, F. Slomka, *Controller Area Network (CAN) Schedulability Analysis with FIFO queues*, Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on. IEEE, 2011.

[18] P. M. Yomsi, N. Navet, R. I. Davis, *Controller Area Network (CAN): Response Time Analysis with Offsets*, Factory Communication Systems (WFCS), 2012 9th IEEE International Workshop on. IEEE, 2012.