Implementing the Continuous Stream Model for Real-Time Control in Linux

Bernardo Villalba Frias
Luca Abeni
Luigi Palopoli
Daniele Fontanelli
DISI - University of Trento
Via Sommarive 5 38123 - Trento
villalba@disi.unitn.it, luca.abeni@unitn.it, palopoli@disi.unitn.it, fontanelli@disi.unitn.it

Abstract

This paper presents the implementation of the Continuous Stream task model (CS) in the Linux Kernel. The CS model is a model of computation for real-time control tasks, whose primary goal is to obtain substantial resource savings without sacrificing the control performance. In this paper, we show the application of the CS model to a robotic application where a mobile robot is requested to follow a line on the ground. The analysis of the data collected from the real experiments confirms the consistency between theory and simulations and the experimental results, verifying the validity and effectiveness of the Continuous Stream approach in a real scenario.

1 Introduction

An interesting trend in the design and development of modern embedded control systems is the increase of the CPU (and sensors) power, which leads to an increased resource sharing, and encourages to run multiple control applications on the same embedded board.

However, running multiple applications on the same CPU/core reduces the determinism of the applications' response times (due to the interference between different applications), while the advanced CPU architectures tend to reduce the determinism of their execution times. These are critical issues given that real-time control applications must be highly predictable in terms of performance.

To enforce this requirement, the conventional approach in real–time control design relies on the combination of a time-triggered model of computation [9], which forces the communications between plant and controller to take place at precise points in time, with the hard real-time scheduling theory [10], which ensures that all the activities are always able to deliver the results at the planned instants.

This approach is efficient and sustainable in terms of resource utilisation only if the computation time of the task does not change too much. However, this assumption is not realistic when the control application uses data coming from complex sensors (such as cameras, RADARS, LIDARS). Under these conditions, the classical hard real-time design approach which allocate computational resources based on the worst-case demand of the application, results in overly conservative choices which could lead to a massive waste of resources and to a drastic reduction in resource sharing.

These issues can be addressed by combining a proper CPU scheduler (based on resource reserva-

tions) with an appropriate task model called *Continuous Stream task model (CS)*. The former guarantees that the different tasks will have timely access to the computational resources, while the latter forces the interaction between computers and environment to take place on precise instants of time, simplifying the construction of a stochastic model for the delays introduced in the control loop.

The recent literature has sought ways to modify the scheduling behaviour, re-modulating the task periods in overload conditions [3, 4] or describing the possible activations of the control task that can be skipped without compromising stability [6].

In [7], the classic idea of the time-triggered approach is considered but with a minimal variant: when a task does not meet its deadline, its execution is cancelled. The solution shows how to combine the flexibility in accepting a variety of possible timing behaviours [5] with the simplification in the system analysis typical of a simple cancellation policy [8].

This paper presents an experimental application and comparison of these solutions within a robotic scenario, where a mobile robot is required to follow a line on the ground. The control task, periodically activated by a timer, compete for the CPU with other tasks. A resource-based scheduler allocates the CPU in presence of simultaneous execution requests. Additionally, the use of the CPU is minimised in an attempt to save resources for other applications.

The paper is organised as follow. Section 2, presents an overview of the implemented Continuous Stream task model. Section 3, provides specific details of the implementation of the task model. Section 4, reports the experiments and obtained results. Finally, Section 5, shows the conclusions and discusses future work directions.

2 The Continuous Stream Model

The model of computation of a real-time task (also referred as "task model") consists of a set of rules to decide: 1) when a job is activated (the job's *arrival time* r_j), 2) when it samples the input (the sampling time s_j), and, 3) when it releases the output (at time v_j).

In this implementation, the controller is considered as a real-time task τ, which is periodically activated after a fixed amount of time called task period T. The control task consists of a stream of jobs J_j with $j \in \mathbb{Z}_{\geq 0}$. Each job J_j is activated at time r_j and finishes at time f_j, after being executed for a time c_j. The job J_j is also characterised by a deadline d_j, that is respected when $f_j \leq d_j$, and is missed when $f_j > d_j$. Moreover, each activation time is also the deadline of the previous job ($d_j = r_{j+1}$ or, equivalently, $d_j = r_j + T$). The response time of job J_j is $\rho_j = f_j - r_j$.

The objective of job J_j is to produce a control action u_j, to be applied to the actuators, based on the sample y_j collected from the plant (and on its past history). The sample y_j is collected at time s_j and the output u_j is released at time v_j. These specific points in time, where sensing and actuation take place, are called *interaction points*.

In the CS model, job activations occur within a limited set of interaction points, which can be chosen as integer multiple of a minimum time granularity P. In contrast to other models proposed in the literature, such activations are not triggered by an absolute time, but by the completion of the previous jobs. This feature ensures that the delay introduced by a job is independent of the delays experienced in the previous ones.

Therefore, the idea of periodic activation is relaxed: when a job finishes before its soft absolute deadline ($f_j < d_j$), then the next job is activated one period T after the job activation. On the other hand, when a job finishes after its soft absolute deadline ($f_j > d_j$), it triggers the activation of the following job. This continuous activation is the reason of the name "Continuous Stream". As mentioned before, the job is actually activated at the next interaction point. The model is described by the following rules:

1. Each job is activated at maximum between the interaction right next the end of the previous job and the previous job's arrival time plus T, $r_j = \left\lceil \frac{max(f_{j-1}, r_{j-1}+T)}{P} \right\rceil P$. For the first activation it is possible to set $r_0 = 0$.

2. If a job J_j experiences a delay response time greater than a threshold $D^{(max)}$, it is cancelled, and a new job J_{j+1} is activated.

3. The delivery of a job's result takes place at the arrival time of the next job ($v_j = r_{j+1}$).

4. The input for a job is collected at the same time the result of the previous job is released

($s_{j+1} = v_j$).

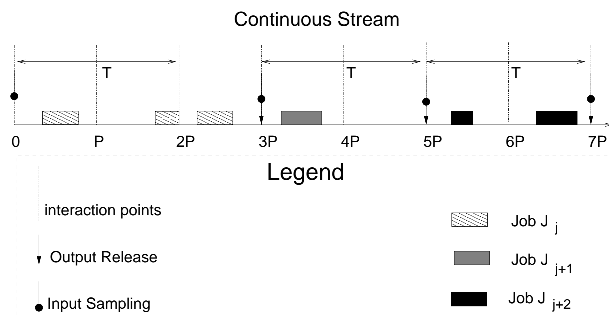

FIGURE 1: *Continuous Stream model: an example*

Figure 1 shows an execution example of the CS model. In the example, the interaction points are defined at regular intervals P and the control task is activated periodically, with period $T = 2P$. The maximum delay allowed is set to $2P$. Job J_j finishes between $2P$ and $3P$, its output is released at time $3P$. At the same time the new input for job J_{j+1} is collected and the job is started. The same happens at time $5P$ for job J_{j+2}.

2.1 The scheduling algorithm

While the Continuous Stream task model can be implemented over any generic real-time scheduler, the analysis of its performance is much easier if the scheduling algorithm provides *temporal protection* (also known as temporal isolation). This property requires that the worst-case temporal behavior of each task does not depend on the other tasks running in the system (in other words, each task is guaranteed to receive a well-defined share of resources, independently of the fluctuations on resource requirements of the other tasks).

Resource Reservations [12] proved to be very effective in providing such temporal protection and have been implemented in different real-time systems using different scheduling algorithms (but have never been implemented in a mainline OS kernel until recently). A reservation is a pair (Q_i, R_i), where Q_i (the *maximum budget*) is the amount of time that the task is allowed to use the resource within every *reservation period* R_i. The ratio $B_i = Q_i/R_i$ represents the fraction of resource utilization dedicated to task τ_i and is often called *resource bandwidth*, or *bandwidth* for short. When scheduling a periodic real-time task, it is often useful to set the reservation period R_i as an integer sub-multiple of the task period T_i ($T_i = NR_i$, where $N \in \mathbb{N}$).

A task τ_i attached to a CPU reservation (Q_i, R_i) can execute for a time Q_i every period R_i with a real-time priority, assigned according to some real-time scheduling policy. The resource reservation algorithm used in this paper is the Constant Bandwidth Server (CBS) [1] (see next section for a description of the algorithm).

The policy used to choose the reservation parameters (Q_i, R_i) depends on the real-time constraints of the task and on its computation requirements. Some previous works [2, 11] show how to assign the reservation parameters Q_i and R_i to control the probability to miss a deadline for task τ_i when the probability distribution of the inter-arrival and execution times are known.

In the case of the CS model, this analysis can be simplified. Only a relative deadline has to be guaranteed (the time between the start of the job and its termination). Given that J_j always starts after the previous jobs have been completed, its finishing time does not depend on the previous jobs (but only on its own execution time) and the analysis can "forget" the history.

In this paper, the reservation parameters Q_i and R_i are statically assigned to tasks and do not change throughout the execution. The reservation period coincides with the interaction points where sensing and actuation take place.

3 Implementation

Since a reservation-based CPU scheduling policy, called SCHED_DEADLINE (which actually implements the CBS algorithm), has just been merged in the Linux kernel, Linux can now be used to implement the CS model described above. This scheduling algorithm is based on dynamic priorities, and in particular on the Earliest Deadline First (EDF) algorithm.

As previously mentioned, SCHED_DEADLINE provides temporal protection between tasks. In order to achieve this goal, each task is characterized by a *runtime* (the budget that have been introduced in previous section) and a *period*. Moreover, the scheduler associates a *scheduling deadline* to each task,

and the ready task having the earliest scheduling deadline is selected for execution. During execution, the runtime is decreased by an amount equal to the time executed, and when a task's runtime reaches zero the task is *throttled* until its scheduling deadline (when a task is throttled, it cannot be selected for execution). At the deadline point, the runtime is refilled, the scheduling deadline is postponed by one period (decreasing the task's priority) and the task exits the throttled state, returning to be schedulable (see the original CBS paper, or the `Documentation/scheduler/sched-deadline.txt` file in the Linux kernel for more details).

As a result, each task is guaranteed to receive a share of the CPU time equal to the ratio between the runtime and the period. Of course, the sum of the fractions of CPU time assigned to each task cannot be higher than the total CPU bandwidth available in the system (usually equal to the number of CPUs).

In this paper, the reservation period has been set to $R_i = 33.33ms$, which is one third of the period of the control task ($N = 3$). Based on [7], it is possible to analytically estimate, the minimum bandwidth B_i that has to be reserved to the tasks in order to ensure the achievement of the stability requirements.

3.1 Platform model

In this implementation, it is considered a set of 4 tasks ($T = \tau_i$ with $i \in \{1, 2, 3, 4\}$) sharing the same computation platform. These tasks are implemented as threads; meaning that they can be managed independently by the operating system scheduler. They are described as follows:

1. *Image capture task:* responsible for interacting with the camera and obtaining the image frames to be processed. The periodic characteristics of this task depends on the frame rate supported by the camera.

2. *Control task:* responsible for implementing the model of computation that establishes the precise time for the next activation of the controller. Additionally, it performs the control of the robot by receiving its position and orientation and producing the control output that guarantees its stability.

3. *Image processing task:* responsible for the processing of the captured images. It runs the image processing algorithm to produce the position and orientation of the robot with respect to the line.

4. *Encoder reading task:* responsible for, periodically, query the encoders to reconstruct the followed path.

Algorithm 1: Continuous Stream

```
void *continuousStream(void *args)
/* Start the image processing */ ;
processImage();
/* Timer for controller's next activation */
*sleep = startTimer(T) ;
while 1 do
    waitActivationAt(sleep) ;
    /* Image processing found a solution */
    if control then
        control = 0 ;
        /* Execute the controller */
        executeControl();
        /* Start the image processing */ ;
        processImage();
        /* Set timer for controller's next
        activation */
        setTimerAt(sleep, T) ;
        /* Current image has no delay */
        delayed_periods = 0 ;
    else
        /* Reached maximum delay */
        if delayed_periods == D^(max) then
            /* Cancel image processing */
            cancelProcess();
            /* Start a new image processing */
            processImage();
            /* Set timer for controller's next
            activation */
            setTimerAt(sleep, T) ;
            /* Reset the delays counter */
            delayed_periods = 0 ;
            control = 0 ;
        else
            /* Set timer for controller's next
            activation */
            setTimerAt(sleep, R) ;
            /* Increase the counter of the
            delays */
            delayed_periods++;
        end
    end
end
```

Based on the description of the task, it is possible to observe that some tasks in T are real-time tasks (e.g., the image processing task), while others are best-effort tasks (e.g., the image capture task). The former have temporal constraints on their execution, while the latter do not receive any kind of temporal guarantees.

In Algorithm 1 it is possible to observe a basic pseudo-code algorithm with the implementation of the CS model.

4 Experiments

The implemented model was tested within an experimental robotic scenario as shown in Figure 2. This robotic scenario represents the path-following problem of mobile robots in which the vehicle forward speed satisfy a reference speed while the controller acts on the vehicle orientation to steer it to the path.

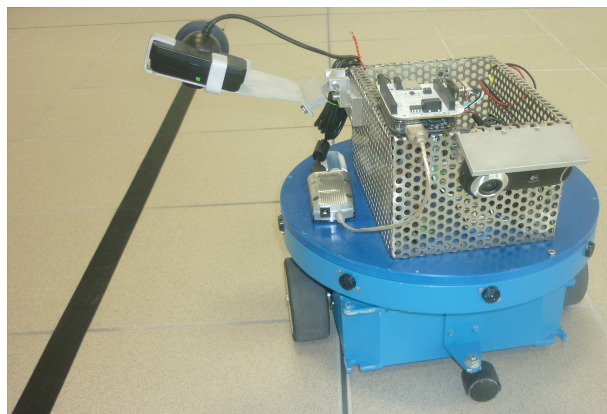

FIGURE 2: *Mobile robot used in the experiments.*

The experimental setup consisted of a straight line placed on the floor creating a track of 12 meters long. The linear velocity was constant and set to 0.6 m/s. The controller has been automatically derived using the Linear Quadratic Gaussian (LQG) systematic design procedure.

In the experimental activities reported, the sampling period has been set to $T = 100$ ms. The number of server periods in the nominal system sampling period was chosen to 3, hence $R = 33.33$ ms. The maximum number of allowed delays $D^{(max)}$ has been fixed to 3. Additionally, a comparison between the CS model and the more traditional Soft Real-Time (SRT) task model [6] has been performed.

As a first step, there was performed a simulation test to determine analytically the minimum CPU utilization (bandwidth) that guarantees the Quality of Control (QoC) requirements of the control task. To accomplish this analysis, it was necessary to estimate two parameters from the data collected during an initial experimental phase: first, the worst case execution time (WCET) of the image processing task, and second, the parameters of the probability density function that best fits the distribution of the computation time.

This initial experimental phase consisted of a series of path-following experiments with a full utilization of the resources (bandwidth of 100%). In total, there were performed 30 experiments with 100% of the bandwidth which resulted in the computation time analysis of more than 6200 samples.

Using these preliminary experiments, it was possible to determine that the probability density function of the computation time was a beta distribution defined in the range [15, 195] ms with a mean value of 25.5246 ms (standard deviation 11.0691 ms). The distribution parameters were $\alpha = 2.6527$ and $\beta = 39.7172$. With these parameters, it was possible to simulate the behavior of the control task and to determine the minimum CPU utilization to sustain the QoC specification.

Table 1 presents the simulation results based on the data obtained from the initial experiments. First of all, it is possible to observe that the CS requires less bandwidth than the SRT. However, under these specific conditions, the variation on the velocity of the robot does not represent a considerable influence in the minimum bandwidth.

Task	Velocity [m/s]				
model	0.4	0.5	0.6	0.7	0.8
CS	20.5	20.8	21.0	21.2	21.4
SRT	29.5	29.7	29.9	30.1	30.2

TABLE 1: *Simulated minimal bandwidth estimation.*

The simulation analysis reports that, for the case of the CS model, the image processing task has to receive 21-22% of CPU utilization to guarantee the QoC requirements. In the case of the SRT model, the minimum CPU utilization required to satisfy the

QoC specifications is 30-31%.

During the experimental phase, and based on the simulation results, there were tested 10 different bandwidths in the range [18% - 36%]. For each bandwidth, 10 trials were performed.

Figure 3 shows the comparison of the trajectory followed by the robot under the CS and SRT models with the same bandwidth. It is possible to note that, at the same low bandwidth, the CS model is able to stabilize the robot allowing it to complete the path; while, in the SRT approach, the robot diverges.

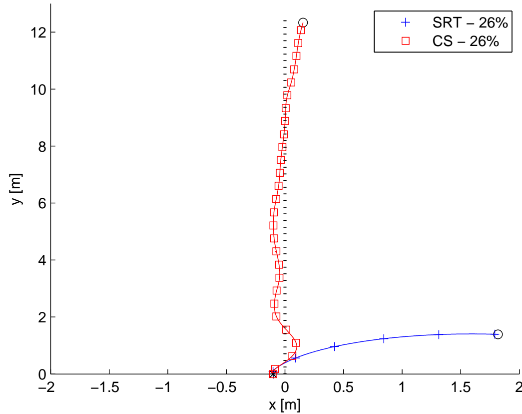

FIGURE 3: *Trajectory comparison at same bandwidth.*

From the *QoC* perspective, the effect of minimizing the bandwidth generates a lower level of performance. However, the stability of the system is still guaranteed. Figure 4 reports the root mean square of the deviation from the desired position. It is possible to observe that the error tends to decrease while the bandwidth increases.

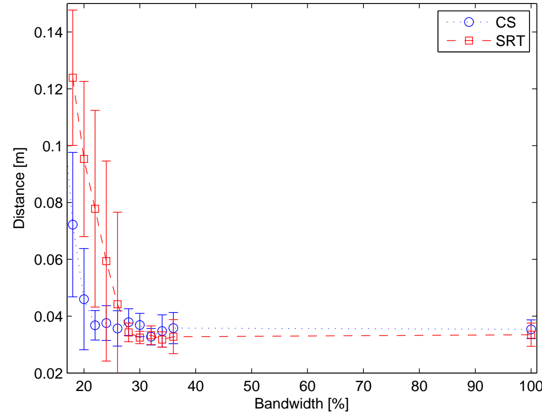

FIGURE 4: *Root mean square of the deviation from the desired position.*

The performance dramatically improves when more than a minimal bandwidth is allocated (22% in the case of the CS model, 28% in the case of the SRT model). The experimental results present a remarkable similarity with the simulation ones, presented in Table 1.

Using KernelShark (a GUI front end to trace-cmd) it was possible to trace the task scheduling of the implemented solution. Figure 5 partially depicts the scheduling behavior. In the few sample periods reported, all the typical behaviors of the CS model are depicted.

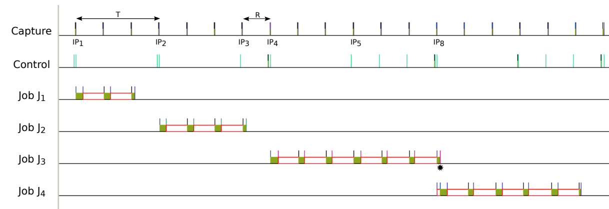

FIGURE 5: *Excerpt of the task set scheduling.*

It is possible to observe that the image capture task presents a perfectly periodic behavior while the control task adjusts its activations according to the rules of the CS model. For example, the first image processing job (Job J_1) is started at the interaction point IP_1 and is finished within its deadline; after a system period T (e.g., at interaction point IP_2), the control task is activated and the controller is able to

release the control output to the motors. This behavior is called "nominal behavior", where the deadlines are respected and the system is stable by design.

When the output is released, a new image processing job (Job J_2) is started. The next activation of the controller is set to one system period T. At interaction point IP_3, the controller is activated but the execution of Job J_2 is not yet finished; hence, the next activation of the controller is changed to the following interaction point IP_4 (e.g., after one reservation period R) and Job J_2 is said to have one reservation period of delay.

At interaction point IP_4, the controller is activated and the execution of the Job J_2 has finished. The controller is able to release the control output to the motors and a new image processing job (Job J_3) is started. Once again, the next activation of the controller is set to one system period T.

In this particular case, it can be noticed that Job J_3 has reached its maximum delay (set to 3 reservation periods) and, at IP_8, the job is cancelled.

After the cancellation, a new image processing job (Job J_4) is started and the next activation of the controller is set. This scheduling behavior continues as expected, based on the rules of the CS model. As mentioned before, in the CS model only a relative deadline is guaranteed and the delay introduced by a job does not affect the execution of the following one.

5 Conclusions

This article showed the implementation (based on Linux and SCHED_DEADLINE) of the Continuous Stream task model. This model allows the simplification of the controller's design and the use of less conservative constraints to impose the stability.

The analysis of the data collected from the real experiments confirms the consistency between theory and simulations and the experimental results, verifying the validity and effectiveness of the Continuous Stream approach in a real scenario. Furthermore, the presented experiments show that new strategies, based on SCHED_DEADLINE, can offer important advantages such as minimizing the resource utilization and guaranteeing the control performance.

As a future work, it is planned to extend the analysis to models without temporal protection between tasks, following novel results on the stochastic analysis for fixed priority schedulers. Moreover, other QoC metrics are being considered, such as minimizing the effect of input noises or including output performance.

References

[1] L. Abeni and G. Buttazzo. *Integrating multimedia applications in hard real-time systems.* In Proceedings of the IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998.

[2] L. Abeni and G. Buttazzo. *Stochastic analysis of a reservation-based system.* In Proceedings of the 15th International Parallel and Distributed Processing Symposium., San Francisco, California, April 2001.

[3] G. Buttazzo, M. Velasco, and P. Marti. *Quality-of-control management in overloaded real-time systems.* Computers, IEEE Transactions on, vol. 56, no. 2, pp. 253266, 2007.

[4] T. Chantem, X. S. Hu, and M. D. Lemmon. *Generalized elastic scheduling for real-time tasks.* Computers, IEEE Transactions on, vol. 58, no. 4, pp. 480495, 2009.

[5] D. Fontanelli, L. Greco, and L. Palopoli. *Adaptive reservations for feedback control.* In Decision and Control (CDC), 2010 49th IEEE Conference on. IEEE, 2010, pp. 42364243.

[6] D. Fontanelli, L. Greco, and L. Palopoli. *Soft real-time scheduling for embedded control systems.* Automatica, vol. 49, no. 8, pp. 2330-2338, 2013.

[7] D. Fontanelli, L. Palopoli, and L. Abeni. *The continuous stream model of computation for real-time control.* In Real-Time Systems Symposium (RTSS), 2013 IEEE 34th. IEEE, 2013, pp. 150159.

[8] D. Fontanelli, L. Palopoli, and L. Greco. *Optimal cpu allocation to a set of control tasks with soft realtime execution constraints.* In Proceedings of the 16th international conference on Hybrid systems: computation and control. ACM, 2013, pp. 233242.

[9] H. Kopetz and G. Bauer. *The time-triggered architecture.* Proceedings of the IEEE, vol. 91, no. 1, pp. 112126, 2003.

[10] F. Liu, A. Narayanan, and Q. Bai. *Real-time systems.* 2000.

[11] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni. *An analytical bound for probabilistic deadlines.* In R. Davis, editor, ECRTS, pages 179188. IEEE Computer Society, 2012.

[12] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. *Resource kernels: A resource-centric approach to real-time and multimedia systems.* In Proc. of the SPIE/ACM Conference on Multimedia Computing and Networking, January 1998.