

Period detection for legacy real-time applications for improving the efficiency in resource scheduling.

Tommaso Cucinotta*, Luca Abeni[†], Luigi Palopoli[†], Fabio Checconi*

{t.cucinotta,f.checconi}@sssup.it {l.abeni,l.palopoli}@unitn.it

*Scuola Superiore Sant'Anna, Pisa, Italy

[†]Università di Trento, Trento, Italy

Abstract—In this paper, the problem is addressed of on-line identification of the optimum scheduling parameters for providing QoS guarantees to multimedia applications. In the proposed approach, the application is required to be modified for using any specific API, but rather its timing behaviour is automatically identified, and appropriate scheduling guarantees are provided by the OS seamlessly, making it particularly suitable for legacy multimedia applications.

The problem is briefly motivated by means of both arguments from the theory of real-time systems, and practical experimental results gathered by using a real-time enhanced Linux kernel. In the proposed approach, an application is traced while it is executing, for the purpose of identifying its characteristic periodicity, if any. Then, an adaptive scheduling reservation is attached to the application, so that the real-time scheduler of the OS may provide it with proper real-time guarantees. Finally, a controller built into the kernel observes the application while it is running, for dynamically adapting the CPU allocation to its continuously changing requirements. Preliminary experimental results are shown, proving effectiveness of the technique in the provisioning of appropriate scheduling guarantees to a widely used multimedia player on Linux. Finally, a detailed roadmap is presented with the possible extensions to the approach.

I. INTRODUCTION

In recent times general-purpose computers have emerged as one of the most effective means to produce, store and distribute multimedia contents. Very frequently personal computers operated by General Purpose Operating Systems (GPOS) are used for video and audio streaming, for editing home-made movies, for video conferencing.

From the perspective of the operating system such applications are very challenging. They are time-sensitive in that the result of the application (its Quality of Service) is enjoyable as long as it is delivered with an acceptable timing. On the other hand the required timing guarantees are not hard as moderate and occasional delays are acceptable if the anomaly is kept in check.

Since such applications are required to share a common computing platform with other applications, a prominent issue is the development of scheduling policies that can be used to ensure their correct timely evolution. A very interesting technology are soft real-time schedulers such as those belonging to the family of algorithms known as *resource reservations* [28]. These algorithms ensure a correct temporal partitioning of the CPU whereby each application is guaranteed a share of its computing power regardless of the behaviour of the

other applications present in the system. This solution has been complemented by adaptive mechanisms to figure out the CPU requirement of time varying or unknown applications and choose the scheduling parameters appropriately [2], [3]. However, an assumption invariably made by such algorithms is that the application is structured as a (typically periodic) stream of jobs and that it makes use of some specialised API available on the underlying OS for: 1) communicating the required scheduling parameters, 2) notifying the start and the termination of each job, so as to trigger the appropriate adaptation logics. This way it is possible for the system to sample the value of some quantities related to the QoS of the application and take corrective control actions (by changing the bandwidth allocated to the task) as needed.

Unfortunately, the set of standard real-time APIs available on a GPOS for building multimedia (or other types of time-sensitive) applications nowadays is merely reduced to the POSIX real-time extensions [13]. These are limited basically to priority-based scheduling, with the addition of mechanisms for dealing with accurate time measurements, timer posting, and dealing with the *priority inversion* problem. While such features may be sufficient for embedded applications, in the realm of General-Purpose applications they are seriously limited, one of the main lacking features being the *temporal isolation* property, ensuring that each application may be designed and run independently from each other.

As a consequence, a wide range of time-sensitive applications exist that deal with their timing requirements in heuristics and quite inappropriate ways, the first one way being by using buffers, which introduce unneeded latencies and decrease the interactivity levels of such applications.

Therefore, in the context of various research projects, a multitude of software architectures have been developed providing well-designed real-time APIs supported by more advanced real-time scheduling capabilities built by modifying the OS at the kernel level. This is for example the case of the architecture developed in the context of FRESCOR European Project¹, or the real-time services which are being designed in the context of the IRMOS European Project².

While open-source applications may be modified in order

¹More information is available at <http://www.fresocr.org>.

²More information is available at <http://www.irmosproject.eu>.

to take advantage of the new real-time OS functionality, for the important class of *legacy applications*, the source-code is not usually available, and they have strong constraints on their lifecycles (subject to commercial policies).

Unfortunately, it is not easy to exploit real-time scheduling guarantees for a legacy application which has not been coded using a real-time API. In fact, the application does not communicate to the OS its scheduling parameters nor the start-time and end-time of its jobs [5]. Therefore, there is no way for the system to associate deadlines to jobs. However, as it will become clear in this paper, it is possible for the OS to infer the necessary scheduling parameters of real-time tasks while they are running up to a certain extent, by recurring to appropriate tracing mechanisms.

The purpose of this work is then to extend the benefits of real-time scheduling to this kind of applications, without imposing any modification to the application themselves.

This is a challenging and multifaceted problem whose solution requires: 1) the ability to correctly infer such important parameters as the activation period of the application, 2) an adaptation of the scheduling parameters to the application ensuring its correct and timely process. We address the first problem by a combination of two technologies: a tracer inside the kernel that extract a time series of events from the kernel and a frequency domain analyser that extracts the most important frequencies from the time series and identifies the fundamental (pitch) frequency using appropriate heuristics. We address the second problem using a feedback scheduler (initially presented in [5]) that observing the evolution of the some scheduling parameters identifies the computation requirements of the task and adjusts the bandwidth accordingly. One of the most point made in this paper is that the effectiveness of the feedback scheduler is greatly magnified by the availability of the task period, reconstructed by the frequency analyser.

The paper is organised as follows.

II. PROBLEM PRESENTATION

In this section, the need for having a period identification mechanism for legacy real-time applications is highlighted, for the purpose of providing to the application a resource allocation as tight as possible to its actual requirements.

After the introduction of background concepts and definitions in Section II-A, the investigation is carried on from the theoretical real-time scheduling perspective in Section II-B. Then, a set of experimental results are shown in Section II-C, which, confirming the theoretical expectations, constitute a fundamental motivation of the presented work.

A. Background and Definitions

In real-time theory, a system is often modelled as a set $\Gamma = \{\tau_i\}$ of real-time tasks³. A very simple yet popular model of a real-time task is the one where a task τ_i is modelled as a stream of *jobs* and is described by a pair (C_i, P_i) : C_i is the worst-case execution time for the individual jobs of τ_i , and

P_i is the minimum inter-arrival time between two consecutive jobs (or the task period in case of periodic tasks). Every job should terminate before the arrival of the next job, and this represents an implicit temporal constraint.

In this work, legacy applications are served by using *resource reservations*, which allow to reserve an amount of time Q^s every period T^s to a task τ (or to a set of tasks implementing an application). This mechanism allows to schedule each task τ_i (or set of tasks) through a couple (Q_i^s, T_i^s) , controlling both the execution rate (the task is allocated a fraction of the CPU equal to Q_i^s/T_i^s) and the responsiveness (the reservation period T_i^s controls the granularity of the CPU allocation) of each application.

The scheduling algorithm used in this work to implement the reservation behaviour is the Constant Bandwidth Server (CBS) [1], which implements CPU reservations by using dynamic real-time priorities (in particular, the CBS uses EDF priority assignment).

The basic CBS idea is to schedule tasks based on their *scheduling deadlines* d_i^s , with d_i^s increased by T_i^s every time that τ_i executes for a time Q_i^s . More formally, the CBS works by maintaining two variables for every reservation: the server budget q_i (used for accounting) and the current scheduling deadline d_i^s (used for assigning a priority to the scheduled task and for enforcement). Such variables are updated as follows:

- When τ_i is created, q_i and d_i^s are initialised to 0;
- When τ_i activates at time t , the scheduler checks if the current scheduling deadline can be used (if $q_i < (d_i^s - t)U_i$), otherwise a new scheduling deadline $d_i^s = t + T_i^s$ is generated and q_i is recharged to Q_i^s ;
- While task τ_i executes, the server budget q_i is decreased as $dq_i = -dt$ (**accounting rule**);
- When the budget is exhausted ($q_i = 0$), it is recharged to Q_i^s and the scheduling deadline is postponed ($d_i^s = d_i^s + T_i^s$) (**enforcement rule**).

Summing up, when scheduling a legacy application (composed by one or more tasks) through a CBS (Q_i^s, T_i^s) , the problem is to infer reasonable values for Q_i^s and T_i^s that allow to serve the application so that it can meet its timing constraints.

B. Period and Budget Adaptation

If the WCET C_i and the period P_i ⁴ of a real-time task τ_i are known, the traditional approach to reservation-based scheduling exploits such knowledge to set $T_i^s = P_i$ and $Q_i^s = C_i$ so that all the task's deadlines are guaranteed to be respected

However, for a legacy real-time application, the enclosing reservation providing scheduling guarantees may not necessarily know the exact period. Therefore, it is not obvious anymore what is a correct budget allocation, that allows the reserved real-time task to respect all of its deadlines.

If a reservation (Q_i^s, T_i^s) is used to serve a single task τ_i , it is possible to investigate the relationship between (C_i, P_i) ,

³The term *task* is used to denote either a process (owning a private memory space) or a thread (sharing the memory space with other threads).

⁴ For more complex set-ups, one may exploit further knowledge like activation jitter or probabilistic modelling of the parameters.

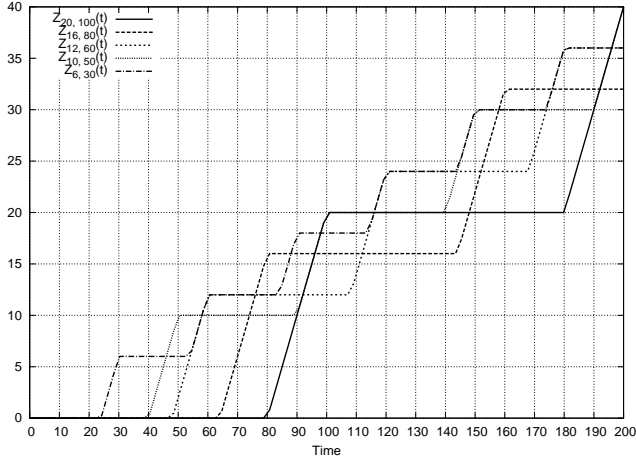


Figure 1. Supply-bound function for different periods, at equal utilization of the resource.

(Q_i^s, T_i^s) , and the QoS provided to the task (in terms of missed deadlines). This analysis can be performed by using the concept of *supply-bound function* $Z_{Q_i^s, T_i^s}(\cdot)$, describing the worst-case amount of time provided by a (Q_i^s, T_i^s) reservation to a task τ_i starting at time 0. Since the reservation abstraction guarantees that Q_i^s units of CPU time are provided to τ_i in a reservation period T_i^s , the worst-case CPU allocation corresponds with the case in which the task is scheduled at the end of the reservation period.

Various supply-bound functions with equal utilization (Q_i^s/T_i^s) but different server periods are shown in Figure 1.

Hence, the resulting supply-bound function is:

$$Z_{Q_i^s, T_i^s}(t) = \begin{cases} hQ_i^s & \text{if } t \in]hT_i^s, (h+1)T_i^s - Q_i^s] \\ t - (h+1)(T_i^s - Q_i^s) & \text{otherwise} \end{cases}, \quad (1)$$

with $h \triangleq \lfloor \frac{t}{T_i^s} \rfloor$.

Based on this definition, one possible test that guarantees that every deadline is respected is based on *Time Demand Analysis* [15]. Such a test checks that the amount of time provided by the reservation to the task before the deadline is enough to serve a job (i.e., $\geq C_i$). In other words, a time-instant exists such that the supply-bound function for the reservation is greater than the worst case execution time of the real-time task:

$$\exists t \in [0, P_i] \text{ s.t. } C_i \leq Z_{Q_i^s, T_i^s}(t) \quad (2)$$

Such test may be used in order to compute the minimum budgets that should be granted to the reservation in order to allow the served real-time task to meet all of its deadlines. Figure 2 (a) reports the minimum budgets needed to correctly schedule a real-time task with a period of $P = 100ms$

⁵Note that a similar concept of supply-bound function is often used in hierarchical scheduling analysis - for example, see [18]. The difference between the function presented here and the one used for analysing hierarchical scheduling systems is that the scenario under investigation is much simpler due to the fact that a single task is being enclosed within the reservation.

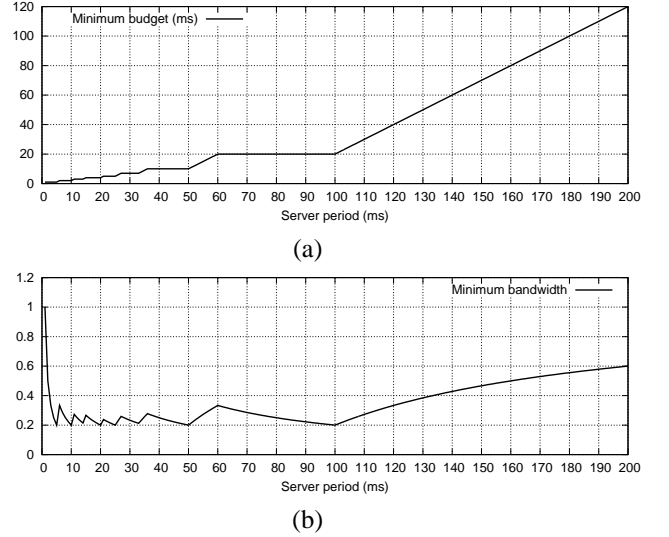


Figure 2. Minimum budget (a) and corresponding bandwidth (b) values required to correctly schedule a $(20ms, 100ms)$ real-time task.

and a WCET of $C = 20ms$ (i.e., a utilisation of 20%). Figure 2 (b) reports the corresponding minimum bandwidth occupied by the reservation. As the plots highlight, setting a completely wrong reservation period (and under the assumption that the feedback-based scheduler manages to identify the corresponding minimum budget), may lead to waste of CPU bandwidth. In fact, if the server period is *smaller* than the task period, then, leaving apart the case for very small server periods which would not be used anyway (due to the unacceptable overheads), it is possible to nearly double the bandwidth requirements of the task, as compared to the actual requirements. If the server period is *greater* than the task period, then the situation goes even worse, because the bandwidth waste grows uncontrolled. On the other hand, the picture also highlights that the best budget assignment is found in correspondence of a server period equal to the actual task period, or an integer sub-multiple of it. However, the choice equal to the task period is the most robust, because small errors in terms of the task period determination are quite well tolerated and lead to the lowest bandwidth wastes. This is also highlighted from Figure 1, showing that, among the supply functions with a utilization equal to the one of the served task (20%), only the ones with a server period equal to the task period (100) or an integer sub-multiple preserve a supply value of 20 time-units in correspondence of the task deadline (100), while the other curves exhibit lower values.

In the previous example, a single task is being considered, but generally a real-time application may be composed of multiple threads of execution with different real-time parameters. When using a single reservation for serving all those threads, the analysis presented above can be extended by reusing concepts from the theory of hierarchical real-time systems [25], [29], [11], [18]. In particular, this requires to use a different definition for $Z_{Q_i^s, T_i^s}(t)$ and to consider a *demand-*

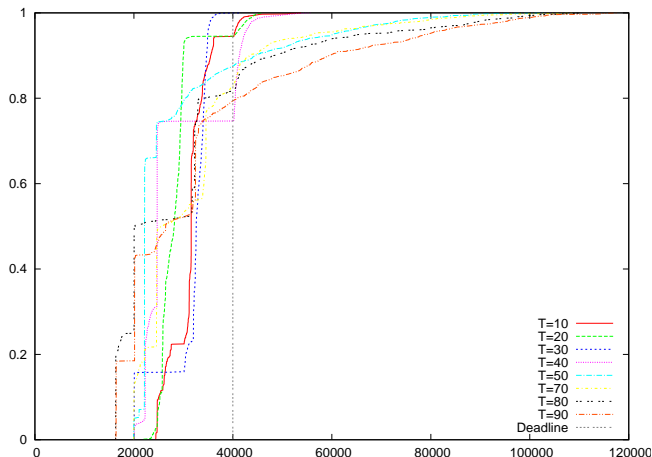


Figure 3. CDF of the response times for a task with period $P = 40ms$ and various reservation periods T^s .

bound function instead of the WCET in Equation 2.

C. An Example

The theoretical analysis presented in the previous subsection is confirmed by some simple examples with two real-time applications (each of them composed by one periodic task) executed on real hardware. To this purpose, each real-time application has been assigned a scheduling reservation with arbitrarily set server periods, while the budget was dynamically computed by the LFS feedback-scheduling algorithm [5] (see Section III-A) so as to allow the application to not exhibit deadline misses.

Figure 3 shows the CDF of the response-time of one of the periodic real-time tasks, when it is controlled by LFS, under various choices of server period. The figure highlights that picking a server period equal to the application period or any integer submultiple attains a quite good performance. In fact, very few deadline misses occur in such cases (a minimum amount of deadline misses is inherent to the way LFS works). However, looking at Figure 4, showing the corresponding dynamic bandwidth allocations made by LFS, it is clear that the best allocation is the one with the server period equal to the application period, corresponding to a much lower bandwidth utilization of the system. On the other hand, the same pictures highlight that, using any other value as server period, may lead to very poor performance in terms of deadline misses.

The bandwidth waste resulting from the use of integer submultiples of the task period as server period, is greater than theoretically foreseen in Section II-B. This was also expected, because the discussion in Section II-B refers to the minimum theoretical budget needed to schedule the real-time task hosted by the reservation, and it does not deal with such issues as how such budget may possibly be found. Due to the particular way LFS works, as it will become more clear in Section III-A, the necessary budget is greatly over-estimated, in such cases.

The experiments shown above highlight that the best results, both in terms of application performance, and of bandwidth

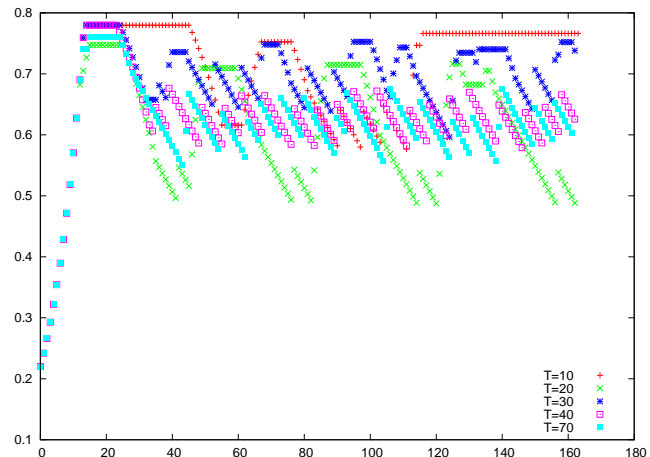


Figure 4. Fraction of CPU bandwidth allocated to a periodic task ($C = 20ms, P = 40ms$) by LFS.

allocated within the system, are achieved when the server period is set equal, or as close as possible, to the task period. This is why in this paper the problem of period detection for legacy real-time applications is investigated.

III. PROPOSED APPROACH

The approach proposed in this paper is summarised in Figure 5. A real-time application is monitored while it is running, by intercepting a set of events generated by it which are of interest from the perspective of inferring its period. Then, the set of times at which these events have been generated is analysed in order to infer the application period, if any. Then, a first rough estimate of the budget needed by the application (with the estimated period) is built, and a scheduling reservation is attached to the application threads. Then, the reservation budget granted to the application is continuously adapted on-line while the application is running, by using the Legacy Feedback Scheduling mechanism. The latter is a very lightweight mechanism built into the kernel scheduler, whose overhead is negligible. From time to time, the period estimation process is repeated in order to gather updated information on the application period. As it will be shown later, this process adds a little overhead to the system which is perfectly sustainable.

The events that are considered as mostly relevant for the purpose of period identification are the ones corresponding to when the application blocks waiting for either a signal or the arrival of a packet from the network or disk, and when it wakes up later. Such events usually occur in correspondence of the call of some blocking system call, like `read()`, `usleep()`, `nanosleep()`, etc.

Therefore, in this preliminary work, a tracing program named `qostrace`⁶ was developed making use of the standard `ptrace()` system call available on Linux, in order to intercept the system calls made by an application at run-time.

⁶For the reader convenience, the program is available at the URL: <http://retis.sssup.it/~tommaso/eng/papers-estimedia09.html>.

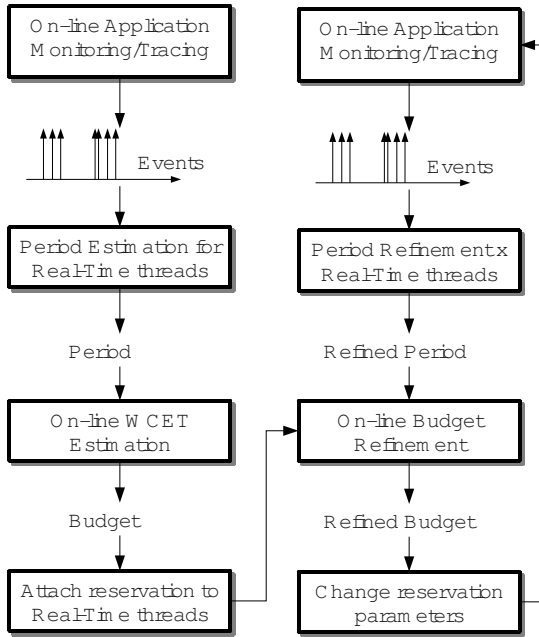


Figure 5. Scheme of the proposed approach.

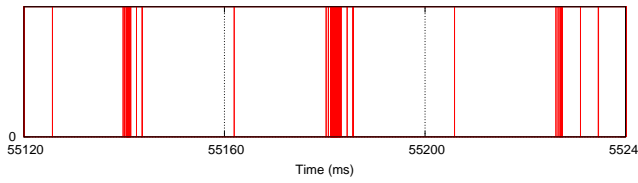


Figure 6. Events generated by mplayer.

This program may be attached to an application while it is running. When attached, in correspondence of each system call entry and exit, the program is suspended, and control is passed to the tracer process, who can perform various kind of inspections on the traced program, then continue its execution. To this purpose of this paper, only the time at which system calls were entered and left was relevant, so the traced program was suspended only for the minimum necessary time (see Section IV-B for overhead measurements).

Figure 6 shows an excerpt of the set of events generated by the `mplayer` software while playing a video at 25fps , i.e., with a period of 40ms . As the picture highlights, every application period there is a conspicuous number of events in correspondence of the activation of each application job.

In the proposed approach, this set of time-stamps is interpreted as a time-continuous signal with a null value everywhere, except at the times in which events were detected, where it exhibits Dirac's Deltas. Then, the first harmonic of this signal is taken as the application period.

In order to do so, the modulus of the frequency-transform of the signal is computed, observed over a sufficiently long time-window, so as to include many application periods. Then, significant peaks of the frequency-transform are searched for, and the smallest value among them is considered as the

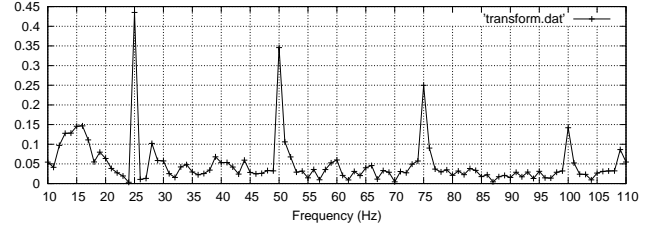


Figure 7. Frequency-transform of the events generated by `mplayer`.

application period.

As an example, Figure 7 plots the frequency-transform obtained for the events collected from the `mplayer` run whose excerpt is shown in Figure 6.

A. Legacy feedback algorithm

To properly serve a time-sensitive task (or set of tasks), the two reservation's parameters Q^s and T^s have to be computed. While a proper reservation period T^s can be estimated by using the techniques presented above, the maximum budget Q^s can be adapted through *feedback scheduling*. For example, the execution time of the tasks can be monitored, and Q^s can be assigned based on the monitored values, or the Legacy Feedback Scheduler (LFS) [5] can be used⁷.

LFS applies feedback scheduling to “unaltered” legacy applications by defining a scheduling error $\epsilon_i = d_i^s - t$ (instead of $\epsilon_i = d_i^s - d_i$, as in adaptive reservations) and by using such scheduling error ϵ_i to adapt Q^s : if $\epsilon_i > T_i^s$ then we can deduct that the application has not been given enough time and Q^s should be increased. A more formal definition of the LFS algorithm follows:

- 1) The control algorithm is executed for *all* time-sensitive tasks with a fixed periodicity T^{sample} ;
- 2) Every T^{sample} time units the scheduling error $\epsilon = (\epsilon_1, \dots, \epsilon_n)$ of time-sensitive tasks τ_i is sampled;
- 3) The reserved times $Q^s = (Q_1^s, \dots, Q_n^s)$ are updated as $Q^s = f(Q^s, \epsilon)$, where $\epsilon = (\epsilon_1, \dots, \epsilon_n)$;
- 4) The scheduling parameters of the different tasks are updated.

Various feedback functions $f()$ have been proposed; in this paper, the simplest one, LFSg, (which is more robust against uncertainties in the tasks periods) is used:

$$Q_i^s = \begin{cases} \alpha Q_i^s & \epsilon_i > T_i^s \\ Q_i^s - \beta & \text{otherwise} \end{cases}$$

IV. EXPERIMENTAL RESULTS

This section reports experimental results made by applying an implementation of the mechanism described above on a real multimedia application.

⁷Notice that Adaptive Reservation cannot be used because legacy applications do not use a real-time API.

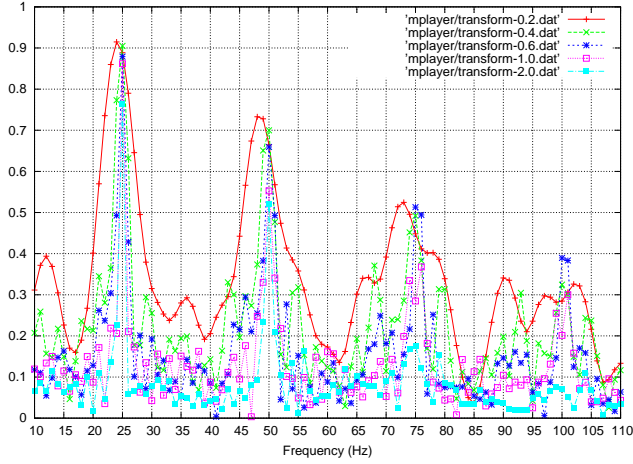


Figure 8. Frequency-transform of the events obtained by tracing `mplayer` at varying tracing durations.

A. Observation Period and Precision

First, the precision of the proposed technique, in relation to the duration of the observation time-period, is analysed. To this purpose, the `mplayer` multimedia player for Linux has been launched multiple times on the same movie, and the proposed tracer was attached at approximately the same relative time-instant from the start of the play, but at varying durations of the observation. Figure 8 reports the obtained frequency-transforms in the various cases of observation durations ranging from 0.2 seconds to 4 seconds. It is clear that, increasing the observation duration, the peaks of the frequency-transform corresponding to the real application frequency ($25Hz$) become much more neat. Moreover, for an observation period of 0.2 seconds or below, the first harmonic is detected almost correctly, with a little error. On the other hand, with observation durations from 0.4 seconds and beyond, the application frequency is detected without mistakes, but the frequency peak becomes more evident and sharp by increasing the observation duration. Note that, with an observation duration of 0.4 seconds (green curve), many frequency peaks are “doubled”, i.e., the frequency peak is dangerously “wide” and “rounded”, so it is possible to perform a little error on the determination of the actual frequency.

Finally, Figure 9 shows the period as automatically detected by the detection algorithm heuristic. As it can be seen, with very short tracing times like $0.2s$, corresponding to barely 5 jobs of the player, the frequency is slightly underestimated, while increasing the tracing time beyond $0.4s$, corresponding to barely 10 jobs, does not seem to lead to any advantages.

Summarising, the preliminary experimental validation conducted over the single-threaded `mplayer` application shows that a tracing time of barely 10 jobs is sufficient for detecting the period of the application with a sufficient precision. This confirms the usability of the methodology sketched out in Figure ??, in which the application is periodically traced in order to detect possible variations in its run-time period.

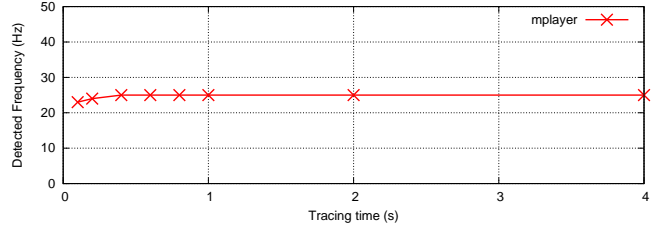


Figure 9. Period automatically detected over the frequency transforms of Figure 8, at varying tracing durations.

Table I
MEASURED OVERHEAD IN TERMS OF CPU %, OBTAINED WITH DIFFERENT TRACERS.

Program	Overhead (CPU %)
<code>strace</code>	6 – 8
<code>qostrace</code>	1

However, in order to verify the feasibility of the approach, it is very important to gather overhead measurements of the introduced mechanism, what is done in the section that follows.

B. Tracer Overhead

A few experiments have been performed in order to gather information on the run-time overhead imposed by the proposed mechanism on the (legacy) real-time applications that are being traced, as well as on the system.

The `strace` program available on Linux already provides the needed tracing functionality, by using the `ptrace` system call, but unfortunately it exhibits a high overhead due to the behaviour of the program, which, after having detected each event, writes a well-formatted and human-readable information row on the standard error program stream.

Therefore, a custom tracer has been developed, exploiting the same principle, named `qostrace`. This program, given the `pid` of the process to trace on the command-line, attaches to it by means of the `ptrace` system call. However, it limits itself to store into an in-memory array the set of events of interest, along with their time of occurrence, then it performs on the data set the required computations.

The overhead measurements are summarised in Table I.

C. Efficiency in Resource Allocation and Period

Some experiments show the relationships between the reservation period T_i^s (set equal to the estimated task period P_i), the accuracy of the CPU allocation performed by LFS, and the QoS achieved by the application. Such experiments have been performed by using an implementation of the CBS in the Linux kernel [4] and playing an MPEG4 stream at $25FpS$ (hence, the player has a period equal to $P_i = 1000/25 = 40ms$) and using LFS to dynamically adapt the amount of CPU time reserved to the player task. The inter-frame times (intervals between the visualisations of two consecutive frames) and the allocated CPU bandwidth $U_i = Q_i^s/T_i^s$ have been measured when LFS uses different reservation periods T_i^s .

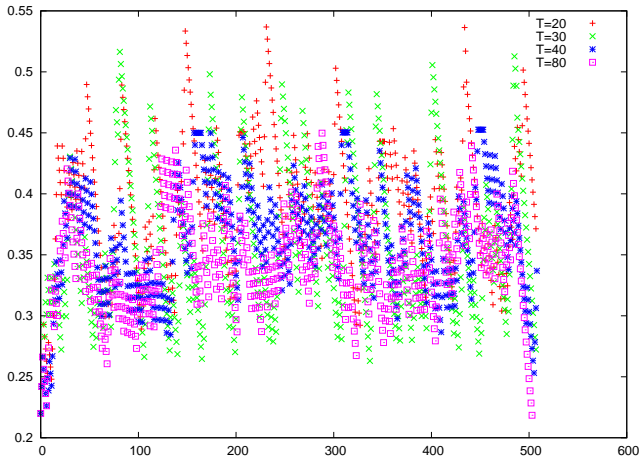


Figure 10. CPU bandwidth allocated to the player as a function of time.

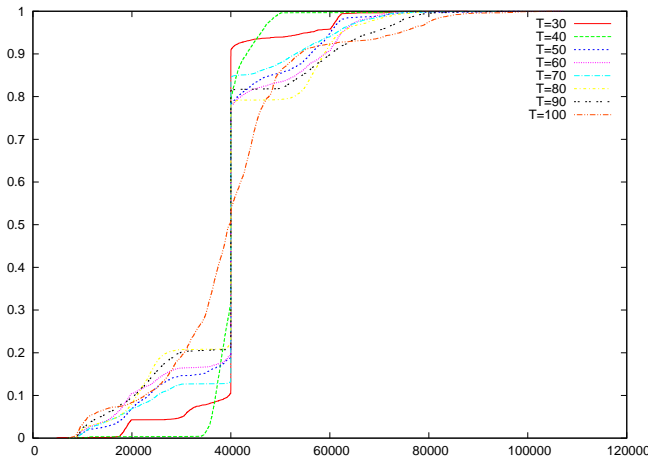


Figure 11. CDF of the inter-frame times for the player.

The fraction of CPU time reserved by LFS to the player is shown in Figure 10. From such figure, it is again possible to notice that if the $T_i^s < 40ms$ then LFS tends to over-allocate the CPU to the task (notice that for $T_i^s = 20ms$ and $T_i^s = 30ms$ LFS arrives to allocate more than 50% of the CPU time to the player, while for $T_i^s = 40ms$ LFS allocates at most 45% of the CPU time to the player). The amount of time allocated to the player if $T_i^s > P_i$ is quite similar to the $T_i^s = P_i = 40ms$ case, hence only $T_i^s = 80ms$ has been displayed in the figure. However, if $T_i^s > 40ms$ the inter-frame times tend to increase: this is visible in Figure 11, which displays the CDF of the inter-frame times. From the figure, it is easy to see that the case with $T_i^s = 40ms$ is the one performing better (in the ideal case, the CDF should be a step going from 0 to 1 at $40ms$).

V. RELATED WORK

The literature on real-time systems in the last few years has produced a good deal of work on how to guarantee the correct timing of a set of time-sensitive activities. It is commonly

believed that traditional real-time scheduling algorithms [19] are not a good match for multimedia applications for they do not offer a sufficient control on the resource allocation. Indeed, with these algorithms, real-time guarantees can be offered assuming the knowledge of such parameters as the Worst Case Execution Time (WCET) and minimum inter-arrival time, which are not easy to know for a large class of applications (e.g., multimedia), let alone if they are legacy. For this class of applications reservation-based schedulers [23], [24], [28], [1] are generally regarded as a better solution. Such algorithms allow one to control the fraction of CPU devoted to each applications but the point remains open as to how appropriately choose the bandwidth if the application requirements are not known and/or change in time.

This problem can be addressed by applying feedback control to real-time scheduling [20], [16], [2], [8], [3]: while the applications execute, their real-time behaviour is monitored and corrective actions are taken changing the scheduling parameters so that specified QoS objectives are met. If additional assumptions can be made on the application, it is possible to use application level feedback such as the one shown in [31]. However, both the used of a specialised API and (all the more) the availability of application level adaptive mechanisms cannot be assumed in the context of legacy applications.

The problem of finding an appropriate allocation for legacy applications is known in the Internet community. In particular in [30] the authors propose an architecture using proxy servers to determine the network requirements of Internet applications.

In the domain of real-time scheduling, there has been some work in dynamically inferring tasks' parameters for legacy applications. For example, BEST [6] tries to infer the tasks' periods by monitoring the times at which the tasks enter the scheduler ready queue. Respect to BEST, the approach presented in this paper separates the scheduling algorithm from the tasks' parameters estimation (allowing to easily combine different reservation-based scheduler with different adaptation mechanisms and period detection heuristics), and uses a more advanced algorithm for detecting periodic tasks.

Other techniques that could possibly be used as a basis for adaptive scheduling of legacy applications have been proposed in the past [7], [17], but to the best knowledge of the authors the first technique developed explicitly to this purpose is in [5], where a feedback scheme for legacy applications was proposed that uses a simple multiplicative/additive scheme to identify the resource requirements by using a coarsely quantised feedback variable.

In this paper this approach is made more effective by complementing the feedback controller with a trace analyser that extracts meaningful information on the task (in our case the period of the tasks) from the time-series of events recorded in the Kernel. This analyser requires the use of two distinct technologies: a tracer component inside the kernel and a spectrum analyser to identify the period of the task. The latter problem is well known in the literature of digital processing of sound signals, where different approaches have been developed to extract the pitch and identify the fundamental frequency [12],

[22]. Such approaches served as a good starting point for our analyser, but we had to adapt them to the analysis of a time-series of events.

As far as the problem of tracing events in the kernel is concerned, there are various mechanisms available, like Linux Tracer Toolkit (LTT/LTTng)⁸, or the more recent `ftrace`⁹ tracer integrated into the mainstream kernel.

VI. FUTURE WORK

The authors plan to work on various improvements to the mechanism presented in this paper, on different aspects: the tracing mechanism, the period detection algorithm, the legacy feedback-based controller, and the support for multi-thread applications. A detailed description of the roadmap follows.

A. Tracing mechanism

The tracing program used in this paper, `gostrace`, uses the `ptrace()` system call, implying the need for suspending the traced process at each occurrence of a relevant event, passing control to the tracing program, then continuing. Even if the overhead incurred by such a mechanism, as measured in the previous section, is sustainable for a large class of systems, it is useful to search for mechanisms which may possibly have a lower impact on the applications that are running.

Therefore, it is foreseen to investigate, in the future, on the use of a kernel-level tracing mechanism registering scheduling events of interest for the traced application, for example the time instants at which the traced process blocks and unblocks. One possibility that we are evaluating is the one of recurring to the use of low-level tracers that already exist for the Linux kernel, like the Linux Trace Toolkit Next Generation (LTTng)[9], [10], the `utrace` [21] and `uprobes` framework [14], or the `ftrace` [10] tracer recently integrated into the mainstream Linux kernel. Such tools can provide a plethora of information on the timing behavior of the kernel and running applications. However, it must be considered that they are developed and maintained mainly as debugging helpers, and are not designed to be actually enabled in a “production” environment. For example, the `sched_switch` tracer of `ftrace`, when configured into the kernel and enabled at runtime, exports by means of the `debugfs` information about all the scheduling events that occur into the system, not only of the traced processes, therefore it is expected that the implied overhead be much higher than strictly necessary. Furthermore, `ftrace` requires administrator’s privileges for being used.

Therefore, while the above mentioned tools (as well as the `ptrace()` system call used in this paper) may be leveraged to build prototypes and proof-of-concepts, it is envisaged in the future the development of a dedicated full-featured kernel-level tracer, or the modification of one of the existing tracing frameworks, for the purpose of overcoming the just mentioned limitations.

⁸More information is available at <http://ltt.polymtl.ca>.

⁹More information is available at: <http://lxr.linux.no/linux+v2.6.30/Documentation/trace/ftrace.txt>.

B. Period detection

The algorithm for period detection presented in this paper is still preliminary. While being effective for the experimental results gathered in this paper, the algorithm needs more extensive validation over a larger class of multimedia applications, comprising single-threaded and multi-threaded applications, and especially commercial legacy software largely used in multimedia streaming, such as QuickTime^(TM) or others.

C. Feedback-based controller

The feedback-based controller used in this paper is also subject to a variety of improvements, the first one being the type of “probe” the feedback-based control loop is based upon. In fact, the boolean information about the CBS deadline having been post-poned or not constitutes a very rough information about how tight the budget in use fits the actual application requirement. Improvements in this direction may be done by exploiting information about the actual execution-time of the reserved threads, as logged by the kernel and available, for example, by means of the `clock_gettime()` system-call via the `CLOCK_THREAD_CPUTIME_ID` clock. Alternatively, specific functions made available by the real-time scheduling framework might be exploited. For example, the AQuoSA [27] real-time scheduler implementing Hard CBS reservations, provides the (`qres_get_time()`) function for the purpose of allowing applications to read how much budget was actually consumed by the set of threads attached to the reservation (as a whole, without any need to query for the individual threads).

With the possibility to directly read the amount of budget actually consumed within the reservation, it would be much easier to decide what budget to assign for the future application jobs. For example, a simple maximum over a moving window of last observed values of consumed budget, or a percentile estimation of the budget consumption distribution, like done in [26], would constitute valuable approached to experiment with.

D. Multi-thread applications

The experimental results presented in this paper are limited to simple applications with a unique evident periodicity, while it is planned to experiment with more complex applications, possibly composed of multiple concurrent threads with possibly different periodicity. For example, a multimedia application with multiple threads, one dedicated to loading the video from the disk (or receiving it from the network), one to video processing and one to audio processing, may possess a different periodicity for the three threads. A good candidate application over which to perform experimental results in this direction would be the Video Lan Coder (VLC)¹⁰.

VII. CONCLUSIONS

In this paper, a mechanism for ... has been presented.

¹⁰More information is available at <http://www.videolan.org/vlc/>.

REFERENCES

- [1] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.
- [2] Luca Abeni and Giorgio Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the IEEE Real Time Computing Systems and Applications*, Hong Kong, December 1999.
- [3] Luca Abeni, Tommaso Cucinotta, Giuseppe Lipari, Luca Marzario, and Luigi Palopoli. Qos management through adaptive reservations. *Real-Time Systems Journal*, 29(2-3), March 2005.
- [4] Luca Abeni and Giuseppe Lipari. Implementing resource reservations in linux. In *Proceedings of Fourth Real-Time Linux Workshop*, Boston, MA, December 2002.
- [5] Luca Abeni and Luigi Palopoli. Adaptive real-time scheduling for legacy applications. In *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2008)*, pages 583–590, Hamburg, Germany, September 2008.
- [6] Scott Banachowski and Scott Brandt. The best desktop soft real-time scheduler. In *Work-in-Progress session of the Real-Time Systems Symposium (RTSS 2001)*, London, December 2001.
- [7] S.A. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003)*, pages 396–407, 2003.
- [8] G. Tao C. Lu, J. Stankovic and S. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Special issue of RT Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, 23(1/2), September 2002.
- [9] M. Desnoyers and M. R. Dagenais. The ltnng tracer: A low impact performance and behavior monitor for GNU/Linux. In *Proceedings of the Ottawa Linux Symposium (OLS 2006)*, pages 209–224, July 2006.
- [10] Mathieu Desnoyers. Ltnng, filling the gap between kernel instrumentation and a widely usable kernel tracer. Linux Foundation Collaboration Summit, April 2009.
- [11] Arvind Easwaran, Insup Lee, Insik Shin, and Oleg Sokolsky. Compositional schedulability analysis of hierarchical real-time systems. In *ISORC*, pages 274–281. IEEE Computer Society, 2007.
- [12] David Gerhard, David Gerhard, and David Gerhard. Pitch extraction and fundamental frequency: History and current techniques. Technical report, 2003.
- [13] IEEE. *Information Technology -Portable Operating System Interface (POSIX)- Part 1: System Application Program Interface (API) Amendment: Additional Realtime Extensions*. 2004.
- [14] Jim Keniston. Uprobes: User space probes. Linux Foundation Collaboration Summit, April 2009.
- [15] John Lehoczky, Lui Sha, and Ye Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proceedings of the Real Time Systems Symposium*, pages 166–171, 1989.
- [16] B. Li and K. Nahrstedt. A control theoretical model for quality of service adaptations. In *Proceedings of Sixth International Workshop on Quality of Service*, 1998.
- [17] C. Lin and S.A. Brandt. Efficient soft real-time processing in an integrated system. In *Work in Progress Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS WIP 2004)*.
- [18] Giuseppe Lipari and Enrico Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2), 2004.
- [19] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.
- [20] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Design and evaluation of a feedback control edf scheduling algorithm. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, AZ, December 1999.
- [21] Roland McGrath. Utrace. Linux Foundation Collaboration Summit, April 2009.
- [22] P. McLeod and G. Wyvill. A smarter way to find pitch. In *Proceedings of International Computer Music Conference, ICMC*, 2005.
- [23] Clifford W. Mercer, Raguanathan Rajkumar, and Hideyuki Tokuda. Applying hard real-time technology to multimedia systems. In *Workshop on the Role of Real-Time in Multimedia/Interactive Computing System*, 1993.
- [24] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda. Processor capacity reserves for multimedia operating systems. Technical Report CMU-CS-93-157, Carnegie Mellon University, Pittsburgh, May 1993.
- [25] Aloysius K. Mok and Xiang (Alex) Feng. Towards compositionality in real-time resource partitioning based on regularity bounds. In *RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium*, page 129, Washington, DC, USA, 2001. IEEE Computer Society.
- [26] Luigi Palopoli, Luca Abeni, Tommaso Cucinotta, Giuseppe Lipari, and Sanjoy K. Baruah. Weighted feedback reclaiming for multimedia applications. In *Proceedings of the 6th IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia 2008)*, pages 121–126, Atlanta, Georgia, United States, October 2008.
- [27] Luigi Palopoli, Tommaso Cucinotta, Luca Marzario, and Giuseppe Lipari. AQuoSA — adaptive quality of service architecture. *Software – Practice and Experience*, 39(1):1–31, 2009.
- [28] Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.
- [29] Saowanee Saewong, Ragunathan (Raj) Rajkumar, John P. Lehoczky, and Mark H. Klein. Analysis of hierarchical fixed-priority scheduling. In *ECRTS '02: Proceedings of the 14th Euromicro Conference on Real-Time Systems*, page 173, Washington, DC, USA, 2002. IEEE Computer Society.
- [30] Charilaos A. Tsetsekas, Sotirios Maniatis, and Iakovos S. Venieris. Supporting qos for legacy applications. In *ICN*, Lecture Notes in Computer Science, pages 108–116. Springer, 2001.
- [31] Clemens C. Wüst, Liesbeth Steffens, Wim F. Verhaegh, Reinder J. Bril, and Christian Hentschel. Qos control strategies for high-quality video processing. *Real-Time Syst.*, 30(1-2):7–29, 2005.