INTERNATIONAL JOURNAL OF PARALLEL, EMERGENT AND DISTRIBUTED SYSTEMS
https://doi.org/10.1080/17445760.2018.1539717

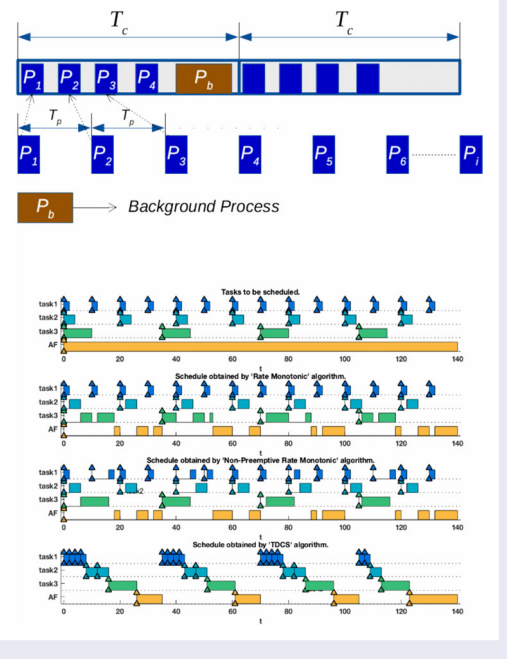
TDCS: a new scheduling framework for real-time multimedia OS

Wei Hu[a,b,c], Tianao Ma[a,b,c], Yonghao Wang [b,c], Fangfang Xu[a,c] and Joshua Reiss[d]

[a]College of Computer Science and Technology, Wuhan University of Science and Technology, People's Republic of China; [b]DMT Lab, Birmingham City University, UK; [c]Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System, People's Republic of China; [d]Centre for Digital Music, Queen Mary University of London, UK

ABSTRACT

The emerging real-time hyper-physical system (CPS), such as autonomous vehicle and live interactive media application, requires time deterministic behaviour. This is challenging to achieve by using the traditional general purpose operating system (GPOS). This paper presents a new design of the real-time operating system (OS) scheduling mechanism called 'time deterministic cyclic scheduling' (TDCS) mainly for live multimedia tasks processing. This new scheduler shares a similar philosophy as classic cyclic execution but with flexibility and dynamic configuration. This hybrid design is based on both time-reserved based cyclic execution and priority-based pre-emptive scheduling for mixed criticality applications. The simulation results show that this scheduling scheme can achieve predictable timing behaviour of task delay and jitter under high CPU utilisation. This shows that the proposed scheme is promising for low latency high-performance multimedia censoring tasks that occur in a periodic manner.



ARTICLE HISTORY
Received 23 November 2017
Accepted 15 October 2018

KEYWORDS
Time deterministic; multimedia; cyclic scheduling; process scheduling; real-time system

CONTACT Wei Hu huwei@wust.edu.cn

1. Introduction

The trend is to use the digital audio workstation (DAW) to perform live music on stage or as the mixer for the front of the house (FOH) system or as the mixer for in-ear monitoring, because of the versatile functionalities, flexible configurations, and expandability that DAW can achieve. On the other hand, audio processing becomes increasingly multifunctional and complex. It is incorporated with the advancement of computation, digital signal processing (DSP) and networking technologies such as high-resolution multichannel audio processing, feature extraction, machine learning and intelligent audio production [1].

Using DAW in a live or interactive environment is bounded by the responsiveness and temporal perception of audio signals. Systems are restrained by interactions of the computational components with the physical processes and are commonly referred to as cyber-physical systems (CPS) [2]. Using DAWs in live or with interaction is an example of CPS.

DAWs are the software systems running on a general purpose operating systems (GPOS). Though it is widely accepted compared with the real time (RT) system. GPOS does not have timing constraints or time criticality. However, there are movements of GPOS towards the mixed criticality system. GPOS has been widely used in various pseudo or soft RT situation, such as multimedia live streaming. An OS that deals with the different levels of time criticality is called the mixed criticality (MC) system [3–5]. It is worth noting the concept of MC implies some trade-off between isolation and integration of resource sharing, whereas systems solely focusing on isolation of tasks are regarded as multiple-criticality systems [5].

Most modern commodity GPOS such as Linux, Windows or MacOS all have some sort of hierarchical scheduling schemes that enable real-time tasks to be executed with minimum jitter and latency. Latest Windows OS has six different priority classes. Mac OS has four different priority bands. Linux by default uses Completely Fair Scheduler (CFS), but it can be configured to use the real-time scheduling policy such as First in First Out (FIFO), Round Robin (RR) and Earliest Deadline first (EDF) with a priority level of 99.

Real-time systems are the systems that not only perform computation with logic correctness but also timing correctness [6,7]. One of the applications of real-time systems is the multimedia system that supports low latency 'live' audio processing. There are classic real-time schedulers such as Cyclic Scheduling, Rate Monotonic Scheduling (RMS) and EDF. However, there is a trend to have RT extension of GPOS such as real-time Linux, since the wide adoption of using open source GPOS in the industry area. Most modern telecommunication platforms and professional live consoles have embedded Linux with RT extension in them to ensure RT performance [8].

However, these modifications may not work for the hard RT system due to the non-deterministic attributes of a file system and device drivers such as the Virtual File System (VFS) framework. That is why traditional RTOS avoids using the file system. To be able to support hard RT in GPOS, one has to rewrite all the file system interfaces and device drivers, such as the proposed work from CMU's RT Mach and RT file system [9].

There are fundamental differences between RTOS and GPOS from a design point of view. RTOS is optimised to be the worst case, whereas GPOS is optimised to be the average case; RTOS targets on predictable scheduling, whereas GPOS targets on efficient scheduling; RTOS is simple executive whereas GPOS has a wide range of services; RTOS tries to minimise the latency, whereas GPOS tries to maximise the throughput. The earlier work of hard RT Linux proposed by Yodaiken et al. was done by replacing all 'cli', 'sti' and 'iret' to the soft interrupt macros and using hardware-triggered interrupts to execute hard RT tasks [10]. However, these hard real-time tasks have no access to Linux kernel services.

Other effort [11] tried to emulate RTOS within GPOS to provide soft RT performance. This paper predicts that the GPOS will be more popular for a soft real-time task such as telecommunication and finance transactions. This research direction is interesting since it is reappearing in the contemporary cloud and virtualisation trend.

In this paper, we proposed a new set of the OS scheduling framework that is called (TDCS) that is specifically tailored for a real-time multimedia system with trading off mechanism of latency and predictable QoE requirements that aims to achieve the predictability of certain tasks using the systematic design approach. The performance evaluation based on simulation between TDCS and RMS is also given to show the pros and cons of it.

2. Background

2.1. Multimedia support on RTOS and GPOS

There were attempts to implement RMS or its modifications such as statistical RMS [12] in GPOS in order to support multimedia applications such as live streaming. RMS is extended into SRMS (statistical RMS) to accommodate for various execution times with average QoS. Then SRMS has been implemented in KURT Linux [13]. The RT Linux work was initially driven by requirements from the live audio community. In 2005, the main target of RT Linux is to support high-performance multimedia and telecommunication applications [14]. Dietrich and Walker also revealed an anecdote regarding an open letter from the audio community to the Linux kernel community that is called 'a joint letter on low latency and Linux' in 2000. This open letter influenced the RT pre-emptive patch developed from 2001.

Since 2008, Linux-based Android OS has gained popularity not only in the smart phone area but also in the embedded world such as set-top boxes and media players. There are development and commercial initiatives to make Android OS to support RT applications including on-demand or interactive media applications [15].

2.2. Live audio processing using OS

Using general operating systems satisfies versatile processing requirements for intelligent audio production [16], such as configurable feature extraction, machine learning and digital signal processing (DSP) tasks. It can process multiple channels with different source sampling rates and flexible routing. This often results in occasional bursting CPU utilisation that is close to full, which often cause the unpredictable jitter of delay of real-time task processing [17] and audio signals [18].

The most concerns of these configurations are the latencies caused by DAWs. In the area of audio engineering, the 'livens' of an audio application dictates the sternness of the input output delay [19–21]. The latency caused by DAWs is due to various signal conversions, processing, buffering, software algorithms and underline OS scheduling.

Using GPOS with software RT extensions reduces the audio processing latency. However, we suspect that using pre-emptive priority scheduling based OS to carry real-time tasks with optimisation can achieve the performance at most of the time, although rarely but it still can fail occasionally [17].

It shows for low buffer settings such as a few audio samples (under 1 ms), the system might only successfully schedule the tasks within time under certain percentage statistically. In addition, the traditional heuristic and ad hoc design approaches have the same predictability problem as described in [7]. The trial and error approach needs to be adopted to find out the minimum latency. This poses the challenge of using traditional scheduling especially when the time and jitter constraints have to be met, whereas the environment has mixed criticality and with sporadic burrstones of tasks. For audio processing, this often results in losing of audio samples occasionally. The human ear is especially sensitive to the timing correctness, even one sample missing will cause audible effects.

2.3. The cross-adaptive model based media processing

In terms of audio, video, and multimedia, due to the real-time nature and specificity of multimedia tasks, the majority of the tasks are periodic. The modern system tends to deal with multiple channels of media sources with possible different rates. When processing the media stream, it often needs to based on timely extracted features from the media to control the process algorithms. This cross-adaptive

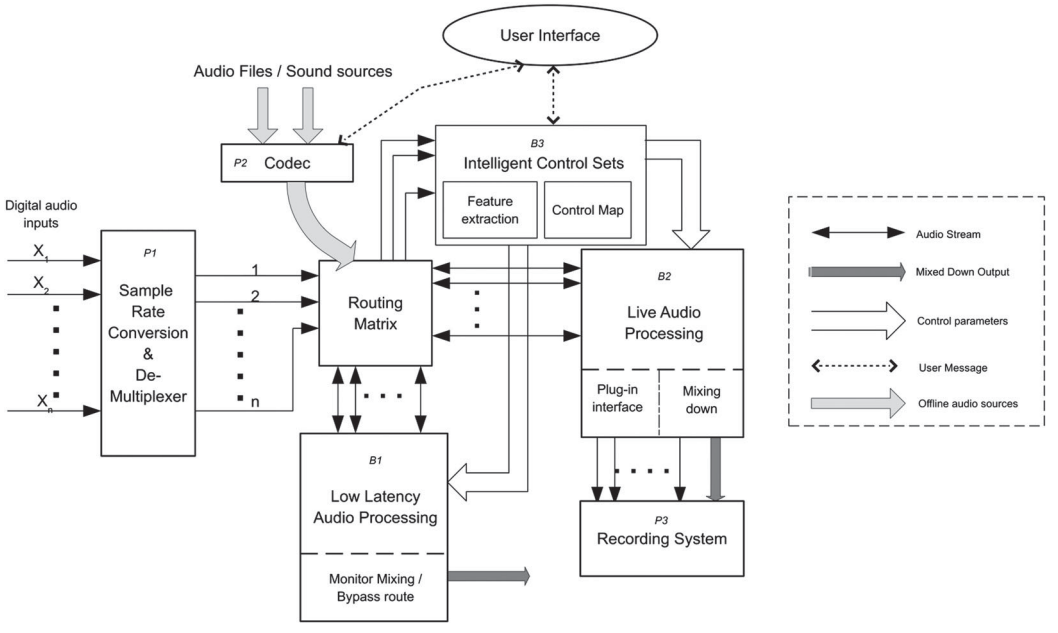


Figure 1. Cross-adaptive audio processing architecture.

model becomes increasingly popular in the AI-based media processing [16]. The cross-adaptive model is expressed in Figure 1.

3. Literature review

Cyclic executive scheduling is one of the earliest schemes that is realised in the real-time operating system between the 1970s and 1980s [22], especially for hard RT tasks. When the system is complex, it is regarded as inflexibile and difficult to maintain. The rate monotonic scheduling (RMS) and the earliest deadline first (EDF) were proposed to overcome these difficulties to assign priority dynamically based on the characteristics of tasks. In the case of RMS, the tasks with the shortest period are assigned the highest priority [23–26].

RMS enabled significant engineering advancement in many areas such as space exploration. On the other hand, the modern forms of cyclic executive based approaches can be found in safety-critical systems as a temporal segmentation mechanism such as 'ARINC Specification 653' that sets standards for avionics RTOS [27] and 'WorldFIP' that defines the Factory Instrumentation Protocol which later becomes a part of the IEC 61158 standard [28]. The cyclic executive approach often is part of the hierarchical scheduling model to ensure temporal segmentation [4,29].

The modern live and interactive multimedia system is an example of a cyber-physical system (CPS) that integrates different inputs and outputs' functions with physical interactions. It has a variety of tasks that require different criticality. In the following, we can look into the characteristics of these live multimedia systems.

3.1. The characteristics of the modern real-time multimedia system

Traditionally, there are two categories of real-time systems: hard real-time system and soft real-time system. In a hard real-time system, the missing deadline of tasks is regarded as a failure, whereas the soft real-time system can have some level of tolerances of missing deadlines. Some textbook regards

the multimedia system as a hard real-time system, due to the strict jitter requirements or human perception of glitch caused by missing the task deadline.

We think the live multimedia system is neither hard real-time nor soft real-time. It lies in between. In a professional audio environment, we would not want to miss any audio/video frames that cause the negative perceptual effects. The jitter of processing single audio or video frame can be tolerated within acceptable user perceptions, using a de-jitter buffer to compensate it at the cost of delay. However, the behavior of jitter should be predictable. In summary, the characteristics of such a system are below:

- It mainly deals with multiple periodic tasks at different update rates with pseudo isochronous tasks pattern.
- It is acceptable to miss the deadline for some tasks but will affect the quality of experiences. Ideally, those tasks that missed deadlines shall not be discarded but still be scheduled at later points.
- It shall provide the trade-off between delay and jitter, using a buffer to mitigate the jitter of the samples. There shall not be any unexpected jitters when the CPU utilisation is close to full.

The traditional cyclic scheduling based approach is difficult to grow and maintain when the number of tasks increases and the periods of tasks are not harmonically related. However, the properties of the multimedia system indicate there are compromises that can be made to adjust the tasks' deadlines within the perceptual tolerance to simplify the system realisation and increase the efficiency of the scheduler.

4. Proposed scheduling scheme

Therefore, we propose a new scheduling scheme that is called TDCS that provides a trade-off between the overall input output delay with other system measures that are based on a traditional cyclic execution based scheduling. In addition, the TDCS is the main part of a hierarchical scheduling scheme for the mixed criticality system.

4.1. Model of rate monotonic tasks

In a real-time system, a task consists of a sequence of jobs. For rate monotonic tasks, we have $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$, where τ is a task set that contains n different tasks. Each task can be defined as $\tau = \{C_i, T_i, D_i, P_i\}$, where

- C_i is worst-case execution time;
- T_i is the period of τ_i;
- D_i is the deadline of τ_i;
- P_i is the priority of τ_i;

For each task, the CPU utilisation is $U_i = C_i/T_i$, so the total utilisation is

$$U = \sum_{i=1}^{\infty} U_{\mathrm{i}} \tag{1}$$

Normally we have $D_i = T_i$. The rate monotonic scheduling (RMS) algorithm assigns the task priority according to the task period. The task with a shorter period has a higher priority. Liu and Layland [24] proved that the sufficient condition of scheduling of RMS is

$$U \leq n(2^{1/n} - 1) \overset{n=\infty}{\longrightarrow} 0.6931 \tag{2}$$

where n is the number of tasks. This condition is sufficient but not necessary based on the tasks that are preemptible. Lehoczky 1989 [30] showed the sufficient and necessary condition of schedulability of RMS with less up bond CPU utilisation but more complex schedulability test formula.

4.2. TDCS algorithm

4.2.1. Temporal segmentation

The foundation of timing correctness this scheduling is based on temporal segmentation is similar to the ARINC 653 standard. The OS is partitioned as independent hyper-periodic segments or major cycle in a time domain driven by an accurate low level system timer. We define this cycle as T_c. The chosen T_c is decided by applications' context.

The selection of the length of segmentation is similar to the problem of selection of hyper-period or size of 'Super frame'. However, we have the flexible architecture of using the segmentation size that practically matches the requirements of the connected system such as USB or networking interfaces.

In theory, the period of temporal segmentation can be the lowest common multiple (LCM) of different task periods. However, this can be very large and impractical to implement. In our design, the period tasks will go through two buffer systems an input 'mapping buffer' and an output 'de-jitter buffer'. The former buffer is used to convert to an arbitrary hyper-period that one wants. The second buffer is to render the tasks as their own source rate.

4.2.2. Hyper-period conversion algorithms

The selection of the hyper-period in TDCS can be flexible. It is not necessary for the LCM of periods of all tasks. As mentioned above it could depend on the application context that is driven by a master clock or the design criteria that need to protect the criticality within a certain time period.

We define LCM of the periods of all tasks as 'major cycle' to be T_L and 'minor cycle' to be T_c. We can have a different way to decide the value of T_c.

For example, T_c can be the longest period of tasks to be scheduled when the CPU utilisation is under the upper bond of RMS schedulability conditions. In the case of the very high CPU utilisation that exceeds RMS schedulability but under 100%, we proposed an 'expanded hyper-period conversion algorithm' to calculate T_c.

Case 1 General hyper-period conversion algorithm: We propose a hyper-period conversion algorithm to convert different T_i to new T_i'. For example, the following algorithm converts the longest T_i as a minor cycle T_c that is normally much shorter than the LCM based hyper-period:

We define the following formula:

$$T_c = \max\{T_i\} \tag{3}$$

$$T_L = LCM\{T_i\} \tag{4}$$

$$K_i = \left\lceil \frac{T_c}{T_i} \right\rceil \tag{5}$$

we have $f_c = \min\{f_i\}$. Increase f_i by Δf_i to be f_i' so that $f_i' = K_i f_c$, where K_i is an integer. We have the new hyper-period T_c and for each task τ_i, we can schedule K_i of them in one hyper-period.

For instance, for 3 tasks with period {10,20,35}, the T_L is 140. $f_i = \{14, 7, 4\}$, we then can find $f_i{}' = \{16, 8, 4\}$ and $K_i = \{4, 2, 1\}$. So in this case, we can use a Task 3 period as the hyper-period. With a task queue, we schedule Task 1 four times, Task 2 twice and Task 3 once within the hyper-period. We introduce a scheduling jitter Δf_i that can be mitigated by the de-jitter task queue. This algorithm creates empty scheduling slots f_i^{empty}, which can be calculated as below:

$$f_i^{\text{empty}} = f_i' - f_i = K_i \times \frac{T_L}{T_c} - \frac{T_L}{T_i} \tag{6}$$

Let $M = T_L/T_c$, we have M number of minor cycles in one major cycle.

Case 2 Expanded hyper-period conversion algorithm: For the CPU utilisation exceeds the upper bond of RMS schedulability, we can alter the algorithm and provide new hyper-period and Task mapping 'As tight as possible,' for utilisation that is close to 100%.

Let total empty slots in one T_c be TS, we have

$$TS = \sum_{i=1}^{N} \Delta f_i \times C_i = \sum_{i}^{N} f_i^{\text{empty}} \times C_i \tag{7}$$

$$T_{ce} = T_c + \frac{TS}{M} \tag{8}$$

In this case, the new minor cycle T_{ce} shall guarantee all the tasks to be scheduled even when the total CPU capacity is 100%. However, for the major cycle the production of T_{ce} and M exceeds the value of LCM. We can further have an adjustment algorithm to make an uneven minor cycle to fit all tasks within one major cycle LCM.

Let j be the index of a minor cycle T_{ce} within the major cycle, so we have

$$T_L = \sum_{j=1}^{M} T'_{ce}(j) \tag{9}$$

Therefore we can have the uneven $T_{ce}{}'(j)$ can be calculated below. Let

$$j_{\text{expand}} = \min\left\{ \left\lfloor \frac{f_i}{K_i} \right\rfloor \right\} \tag{10}$$

$$T'_{ce}(j) = \begin{cases} T_{ce} & j \le j_{\text{expand}} \\ T_{ce} - \sum_{i=1}^{|\tau|} T_j(i) & j > j_{\text{expand}} \end{cases} \tag{11}$$

where

$$T_j(j) = \begin{cases} (j \times K - f_i) \times C_i & j \times K_i > f_i > (j-1) \times K_i \\ K_i \times C_i & j \times K_i - f > K_i \end{cases} \tag{12}$$

where $1 \le j \le M$ and $1 \le i \le |\tau|$.

4.2.3. Periodic tasks mapping schemes.

For the most period task τ_i that happens every T_i, the tasks are scheduled off-line by allocating the τ_i in the cycle T_c. This is done by the task allocation mapping algorithms given below:

(1) Calculate the number of tasks N of τ_i in every minor cycle by$N \ge (T_c/T_i)$. For example, if T_c is 20 ms, T_i is 5 ms, then $N = 4$;
(2) Allocate K timing positions within every $T_c$for τ_i to ensure the same positions for every TL for any τ_i. There are different methods to allocate the positions such as 'even spread,' or as 'tight' as possible. Each of them have different performance effects.
(3) Prepare the multiple low level timers for dispatching the mapped task τ_i and prepare the de-jitter input/output task queue for τ_i.

One of the advantages of TDCS is the predictability and CPU utilisation. We implement the TDCS scheduling algorithm with the 'As tight as possible' tasks mapping scheme and carried out the test. The various results are compared with classic RMS and Non-Preemptive RMS (NP-RMS). Next, we show the performance of TDCS in comparison with RMS and NP-RMS (Figure 2).

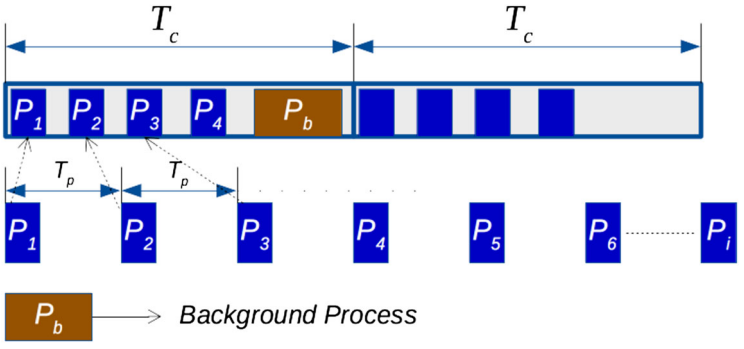


Figure 2. Periodic tasks mapping.

Table 1. Simulation tasks sets and CPU utilisation.

Task Set	Details	Utilisation
Set 1	Task 1 = {2,10,10}	68.57%
	Task 2 = {4,20,20}	
	Task 3 = {10,35,35}	
Set 2	Task 1 = {2,10,10}	82.86%
	Task 2 = {4,20,20}	
	Task 3 = {15,35,35}	
Set 3	Task 1 = {2,10,10}	97.14%
	Task 2 = {4,20,20}	
	Task 3 = {20,35,35}	

4.3. Compare TDCS with non-preemptive RMS and RMS

It is worth to compare TDCS with classic fixed priority scheduling algorithms. For multimedia applications, the absolute deadline is not essential. The non-preemptive RMS (NP-RMS) [31,32] is the RMS scheduling algorithm without preemption which reduces the system complexity. NP-RMS is suitable for multimedia applications where hard RT is not essential. We compare TDCS with both RMS and NP-RMS based scheduling policies. We implemented TDCS scheduling in TORSCHE [33].

4.3.1. Schedulability simulation

We define three task sets that represent the load of CPU from sparse to dense. The task sets used for simulation is described in Table 1. Set 1 is designed so that all three schedulers can successfully schedule all the tasks. Set 2 is designed to make NP-RMS fail to schedule, whereas RMS and TDCS can. Set 3 is designed to simulate so that the CPU is heavily loaded so that both RMS and NP-RMS will miss some deadlines (Table 1).

4.3.2. Simulation results

Figure 3 shows the simulation results of Set 1 which has an average CPU utilisation of 68.57%. The top sub figure shows the three original tasks defined in Set 1. Task 1 has the highest frequency hence will be assigned to the highest priority in the RMS scheduling policy. The P_b process is the background non-RT tasks that can be fit into the gap of RT tasks. The second sub-figure of Figure 3 is the task map of RMS, which shows that some of task 2 and task 3 are delayed or interrupted but can finish within the deadline D_i. The third sub figure of Figure 3 shows the task map of NP-RMS scheduling. For NP-RMS, none of the task can be interrupted, so even the highest priority tasks in task 1 have shown some delays. The bottom sub figure in Figure 3 shows the task map of TDCS. We use the 'as tight as possible' tasks' mapping scheme to maximise the capacity of the system. It shows a clear pattern within the minor cycle and major cycle.

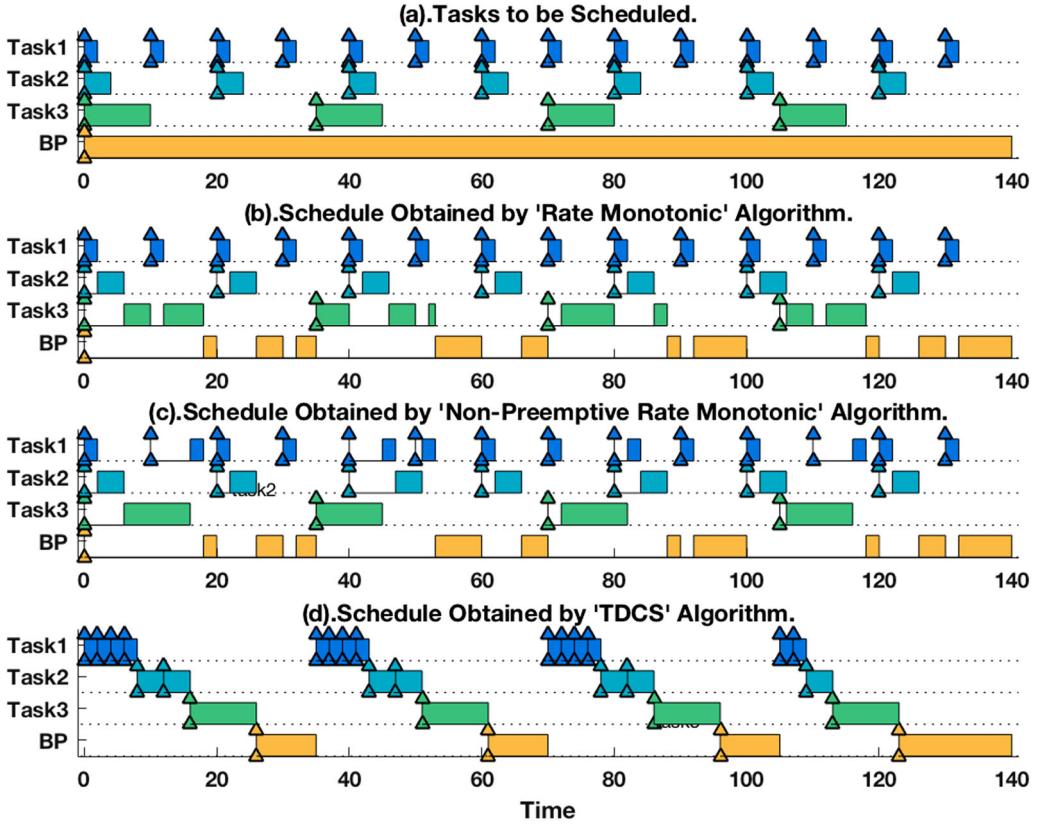


Figure 3. Simulation results for set 1.

Figure 4 shows the results of Set 2 where the average CPU utilisation is 0.8286: In this case, the non-preemptive scheduling algorithm cannot schedule all the tasks. Some tasks in Task1 get lost such as the 2nd 5th, 9th and 13th tasks in Task 1. Figure 5 shows the results of Set 3 which has an average CPU utilisation of 97.14%. In this case, both non-preemptive RMS and RMS scheduling algorithms cannot schedule all the tasks. But with the expanded hyper-period conversion algorithm and 'as tight as possible', the TDCS can successfully schedule all the tasks.

4.3.3. Jitter of individual delay simulation

Figure 6 shows the individual task starting time (a) and the delay between the time of the tasks actually starting execution and the time when the tasks are released (b). Figure 6 is based on Set 1 and compares between TDCS with NP-RMS. Figure 7 is also based on Set 1, but shows the results between TDCS and RMS. The negative value of TDCS delay in the figures is because we buffered TDCS tasks and condense the 'future' tasks together as often done in frame-based audio processing. It shall be added with an offset of buffering delay.

RMS and NP-RMS perform very well if the tasks' loading is not very high. The simulation shows the 'As tight as possible' tasks mapping of TDCS that is geared towards maximising CPU utilisation. Therefore, TDCS shows some fluctuations of tasks delay in comparison with the tasks' release time. With the de-jitter buffer, this effects will be alleviated. We will conduct in-depth research on how to reduce buffer.

4.3.4. Overall task throughput simulation

In this simulation, we create random tasks sets for all three cases TDCS, RMS and NP-RMS. The random tasks sets have three RT takes making up the CPU utilisation range from 60%, 65%, . . . , until 100%.

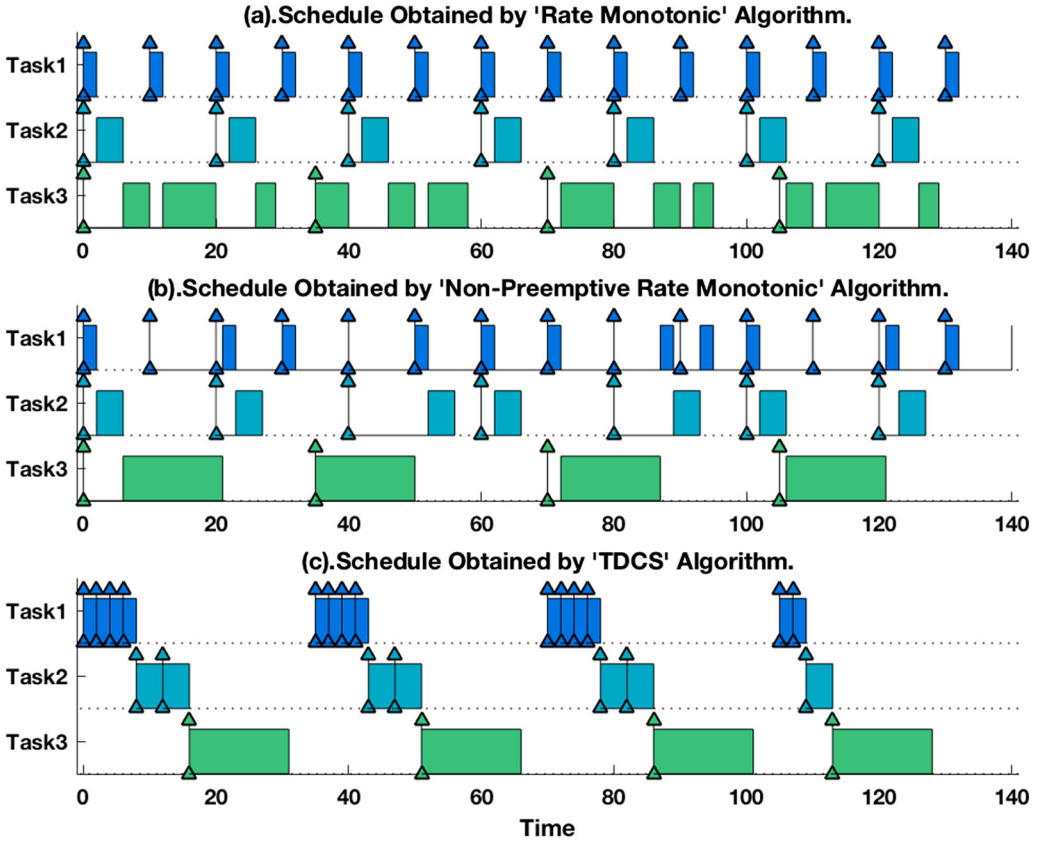


Figure 4. Simulation results set 2.

Each case, we created 100 random sets. The percentages of RT tasks that can be executed successfully are plotted against CPU utilisation for all three different scheduling algorithms. The result is shown in Figure 8. It clearly shows that our proposed TDCS scheduling and 'Expanded hyper-period conversion algorithm' can handle the task load up to 100%.

4.4. Using TDCS in the MC system

In this section, we briefly discuss how to use TDCS in a mixed criticality system or how to incorporate TDCS with GPOS as RT extension.

4.4.1. Slack stealing

The new scheduling framework shall support the slack stealing concept. Even the multiple periodic tasks would not occupy 100% of the CPU time. The background tasks P_b with a lower priority than multimedia periodic tasks τ_i can be scheduled in the slack time of τ_i. However, they can be interrupted by τ_i that has a higher priority and driven by a pre-set low level timer. This concept is demonstrated in Figure 3.

4.4.2. Hierarchy priority scheme

The system can be designed with three tiers of priority ring. The inner ring has a higher priority. In the inner most is the Tier 0 tasks that assigns to the most emergency tasks such as manually terminate the programmer. The most multimedia periodic tasks will be given tier 1 priority. These tasks will be

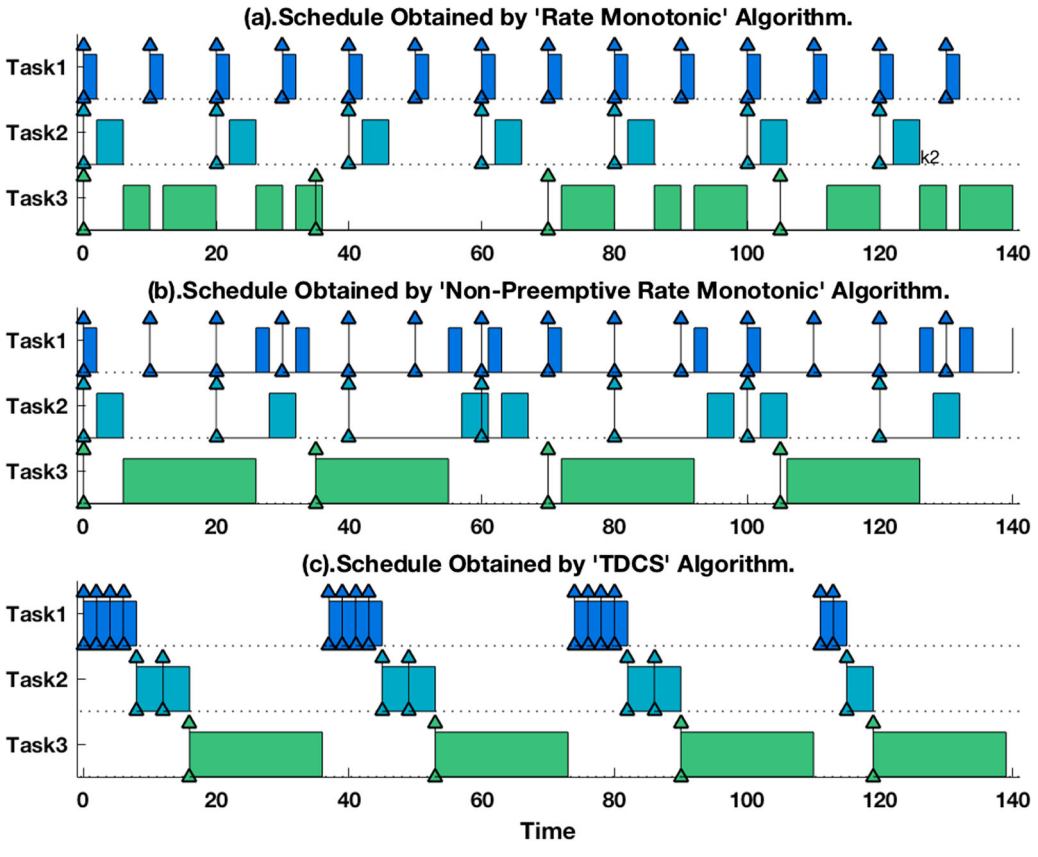


Figure 5. Simulation results 3.

statistically scheduled off-line by TDCS before the task flow is initiated. The background tasks P_b are assigned as tier 2. These can be based on classic pre-emptive scheduling and are scheduled within the slack time of tier 0 and tier 1.

- Tier 0 emergency tasks – the adjustable system timer drives the major cycle and dispatches the periodic tasks belonging to this tier.
- Tier 1 periodic task – multimedia live tasks. Audio samples or frames, video frames, period check and control message.
- Tier 2 pre-emptive background processes.

4.5. Advantages and disadvantages of TDCS

There are a few advantages of integrating TDCS into the current system, especially in the mixed criticality system or as RT extension of GPOS.

(1) Time deterministic for multimedia periodic tasks. The accurate delay can be estimated in advance.
(2) Flexible hyper-period conversion schemes that can create different hyper-periods for different application contexts.
(3) Support high CPU utilisation up to 100% with the selectable task mapping scheme.
(4) Simplified schedulability test, which enables the pseudo on-line scheduling mechanism.

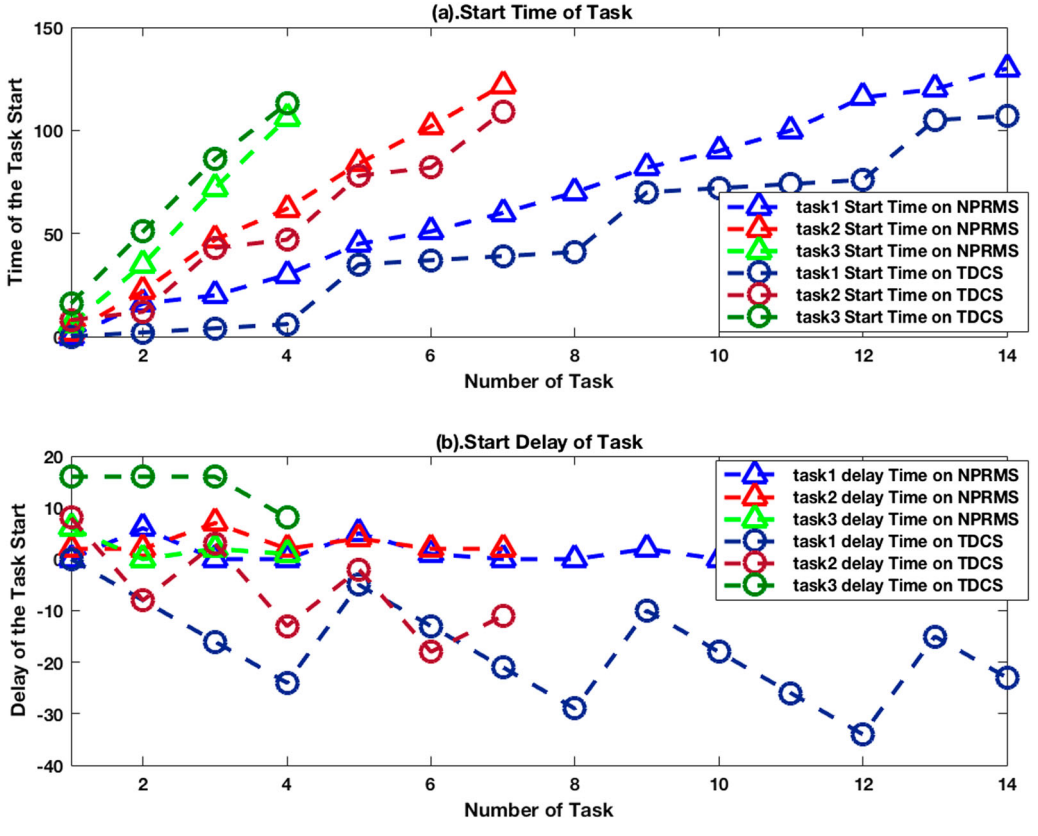


Figure 6. Task delay and starting time TDCS vs NP-RMS.

(5) Efficiency: The multimedia tasks can be driven by low level system timer not the software interrupts. It has fewer context switches.

One of the disadvantages of TDCS is similar to classic RMS and EDF that is to rely on worst-case execution time (WCET) of proper off-line planning. If the WCET value is not accurate that will affect the overall schedulability. TDCS also performs worse than RMS in terms of suitability for hard real-time tasks, because TDCS provides flexibility of re-allocation of tasks in the time line. It introduces the execution jitter; however, the jitter is predicable and can be managed.

4.5.1. Contribute to low latency

Using TDCS to schedule low latency audio tasks shall not allow the unexpected jitter and loss of audio frames to happen in this case by providing full schedulability under 100% CPU load. The system shall provide certainty of whether it can accept more tasks or not. It shall prevent the uncertainty of scheduling latency performance when large high priority tasks are presented in the system.

5. Conclusions

5.1. Summary

In this work, we present a new TDCS scheduling framework for real-time multimedia applications such as low latency audio processing. The design of TDCS is based on the classic cyclic executive concept

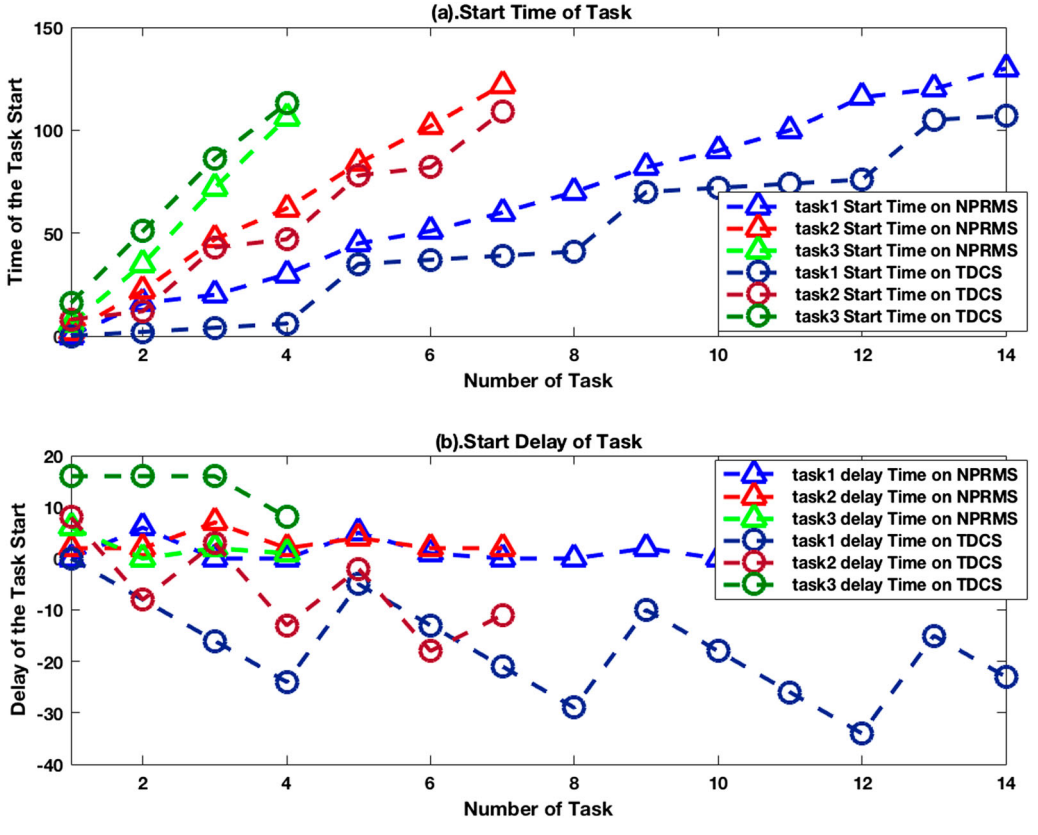


Figure 7. Task delay and starting time TDCS vs RMS.

but with more flexible allocation of tasks and hyper-period design. With ever increased CPU processing power, the TDCS is designed with possibilities of integration of the MC system and GPOS in mind.

The simulation shows TDCS has comparable performance as classic RMS scheduling, especially TDCS is flexible to use different hyper-periods that trade-off with re-allocation of tasks. In addition, TDCS can achieve high CPU utilisation of RT tasks without loss of executions of tasks. That is important for heavy loaded multimedia processing.

5.2. Further work

There are many areas of this work that can be further explored. One interesting work in theoretical aspects might be to have a generic mathematical model for generating the arbitrary length of hyper-periods that is optimised towards different measures such as minimise jitter, delay, or tasks buffer. One of the main direction of this work is to actually put this in use of GPOS such as the Linux system, so in the next step, we will try to integrate TDCS into the current Linux kernel and provide a feasible interface to applications. It might result in some compromise of the original design and practical alteration of the mechanism.

For developing the TDCS framework, such as TDCS use different scheduling modes, more compact scheduling or more smoother scheduling, further in couple with current computer science trends. It is especially interesting to consider TDCS in virtualisation and cloud computing since most future media play out will be cloud based along with portable smart clients. The mapping of the tasks of different multimedia data on cloud computing and the mobile device is an interesting challenging question. The research by Qiu and Zhao [34] provide a promising model that might be considered to

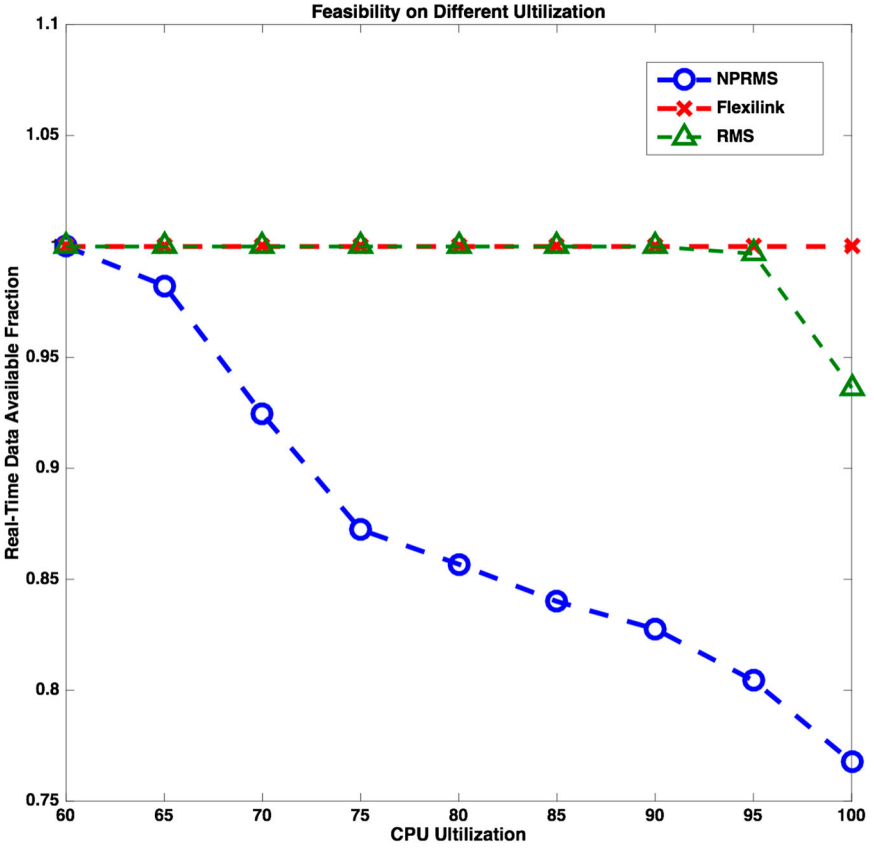


Figure 8. Task throughput vs CPU utilisation.

integrate with TDCS. For a large heterogeneous memory structure that stores the media files, Gai et al. [35] demonstrate a good approach to move TDCS forward in the cloud computing area.

As Lee [2] mentioned, may be the biggest challenge for real-time in the CPS system is the absence of time abstract from different lower layers. That is reflected as the difficulty of estimation of WCET in many cases. For multimedia processing, lots of tasks are DSP based. It might be worth to see how modern DSP acceleration mechanism can be accurately timed and reported for upper layer scheduling algorithm such as TDCS.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by National Natural Science Foundation of China: [Grant Number 61602350].

ORCID

Yonghao Wang http://orcid.org/0000-0002-4924-2508

References

[1] Reiss JD. Intelligent systems for mixing multichannel audio. International Conference on Digital Signal Processing. Corfu: IEEE; 2011. p. 1–6.

[2] Lee EA. Cyber physical systems: Design challenges. IEEE Symposium on Object Oriented Real-Time Distributed Computing. Orlando (FL): IEEE Computer Society; 2008. p. 363–369.
[3] Niz DD, Lakshmanan K, Rajkumar R. On the scheduling of mixed-criticality real-time task sets. Real-Time Systems Symposium, 2009, RTSS. Washington (DC): IEEE; 2009. p. 291–300.
[4] Kelly OR, Aydin H, Zhao B. On partitioned scheduling of fixed-priority mixed-criticality task sets. IEEE, International Conference on Trust, Security and Privacy in Computing and Communications. Changsha: IEEE Computer Society; 2011. p. 1051–1059.
[5] Burns A, Davis R. Mixed criticality systems – a review. York: Department of Computer Science, University of York, Tech. Rep, 2013. p. 1–69.
[6] Stankovic JA. Misconceptions about real-time computing: a serious problem for next-generation systems. Computer. 1988;21(10):10–19.
[7] Buttazzo GC. Hard real-time computing systems: predictable scheduling algorithms and applications. Springer Berlin: Science & Business Media; 2011.
[8] Leroux PN. RTOS versus GPOS: what is best for embedded development? Cupertino (CA): QNX Software Systems Ltd; 2005.
[9] Molano A, Juvva K, Rajkumar R. Real-time file systems: guaranteeing timing constraints for disk accesses in rt-mach. Real-Time Systems Symposium, 1997. The IEEE Proceedings. San Francisco (CA): IEEE; 1997. p. 155–165.
[10] Yodaiken V. The RTLinux manifesto. Raleigh (NC): Proc of Linux Expo; 1999.
[11] Adelberg B, Garcia-Molina H, Kao B. Emulating soft real-time scheduling using traditional operating system schedulers. Real-Time Systems Symposium; 1994. San Juan: IEEE; 1994. p. 292–298.
[12] Atlas A, Bestavros A. Statistical rate monotonic scheduling. IEEE Real-Time Systems Symposium. Santa Barbara (CA): IEEE Computer Society; 1998. p. 123.
[13] Atlas A, Bestavros A. Design and implementation of statistical rate monotonic scheduling in KURT Linux. Phoenix (AZ): Real-Time Systems Symposium; 1999. IEEE; 2002. p. 272.
[14] Dietrich ST, Walker D. The evolution of real-time linux. Rtl Workshop; 2005.
[15] Kalkov I, Franke D, Schommer JF, et al. A real-time extension to the Android platform. International Workshop on Java Technologies for Real-Time and Embedded Systems. Copenhagen: ACM; 2012. p. 105–114.
[16] Reiss JD. Intelligent systems for mixing multichannel audio. International Conference on Digital Signal Processing. Corfu: IEEE; 2011. p. 1–6.
[17] Williams C. Linux scheduler latency. Raleigh (NC): Red Hat Inc; 2002, 3.
[18] Wang Y, Stables R, Reiss J. Audio latency measurement for desktop operating systems with onboard soundcards. London: Audio Engineering Society Convention; 2010.
[19] Lester M, Boley J. The effects of latency on live sound monitoring. 2007.
[20] Farner S, Solvang A, Sæbø A, et al. Ensemble hand-clapping experiments under the influence of delay and various acoustic environments. Néphrol Thérap. 2009;57(12):268–269.
[21] Chafe C, Cáceres JP, Gurevich M. Effect of temporal separation on synchronization in rhythmic performance. Perception. 2010;39(7):982.
[22] Baker TP, Shaw A. The cyclic executive model and Ada. Real-Time Syst. 1989;1(1):7–25.
[23] Sha L, Abdelzaher T, Erik årzén K, et al. Real time scheduling theory: a historical perspective. Real-Time Syst. 2004;28(2–3):101–155.
[24] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard-real-time environment. Readings in hardware/software co-design. Norwell (MA): Kluwer Academic Publishers; 2001.
[25] Sha L, Rajkumar R, Sathaye SS. Generalized rate-monotonic scheduling theory: a framework for developing real-time systems. Proc IEEE. 1994;82(1):68–82.
[26] Sha L, Goodenough JB. Real-time scheduling theory and Ada. Los Alamitos (CA): IEEE Computer Society Press; 1990.
[27] Sławomir S. ARINC specification 653 based real-time software engineering. e-Informatica Softw Eng J. 2011;5(1):39–49.
[28] Almeida L, Tovar E, Fonseca JAG, et al. Schedulability analysis of real-time traffic in WorldFIP networks: an integrated approach. Trans Ind Electron IEEE . 2002;49(5):1165–1174.
[29] Hangan A, Sebestyen G. Cyclic executive-based method for scheduling hard real-time transactions on distributed systems. IEEE International Conference on Intelligent Computer Communication and Processing. Cluj-Napoca: IEEE; 2011. p. 441–444.
[30] Lehoczky J, Sha L, Ding Y. The rate monotonic scheduling algorithm: exact characterization and average case behavior. Rate monotonic scheduling algorithm: Exact characterization and average case behavior. 1989. p. 166–171.
[31] Park M. Non-preemptive fixed priority scheduling of hard real-time periodic tasks. International Conference on Computational Science. Kuala Lumpur: Springer; 2007. p. 881–888.
[32] Nasri M, Brandenburg BB. Offline equivalence: a non-preemptive scheduling technique for resource-constrained embedded real-time systems (Outstanding Paper). Real-Time and Embedded Technology and Applications Symposium. Pittsbburgh (PA): IEEE; 2017.

[33] Sucha P, Kutil M, Sojka M, et al. TORSCHE scheduling toolbox for Matlab. Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control. IEEE; 2009. p. 1181–1186.
[34] Gai K, Qiu M, Zhao H, et al. Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. J Network Comput Appl. 2016;59(C):46–54.
[35] Gai K, Qiu M, Zhao H. Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing. IEEE Trans Cloud Comput. 2016;PP(99):1–1.