Real-Time Scheduling Algorithms and Battery Consumption of Mobile Devices

Raquel Elizondo*, Martín Flores*, Jose Salazar*, Oscar Rodríguez* and Nelson Méndez*
School of Computer Science
Instituto Tecnológico de Costa Rica, Cartago, Costa Rica
**Email*: {rackelelizondo, mfloresg, bimbosalazar, oscar.rodar, n.mendezmontero}@gmail.com

***Abstract*—One of the most increasing areas in application development is real time applications, as time goes by and technology develops more powerful devices the applications are now requested by users as real time applications, extended reality applications, and more complex applications such as online banking and more that requires more complex implementations every time. As much as we as users like these new applications and the new possibilities we have with them, there is always a concern regarding this kind of applications in mobile devices: the energy consumption. For this applications to run and perform as expected, a considerable amount of energy is needed, for these applications the constant communication with peers and/or main services is essential, and for that live interaction the device needs to spend more energy than a plain classic application, specifically for jobs that the processor executes periodically to keep the live interaction as expected. There are some approaches for this problem that involve designs of algorithms for scheduling these kind of jobs with the objective of saving energy, or at least spend it wisely. In this paper we discuss some of the algorithms that have been proposed to mitigate this issue and keep the user experience the best possible by using battery energy in a smart way but still guaranteeing a very good performance of real time applications.**

I. Introduction

NOWADAYS real-time applications are taking over mobile devices more than ever and, although this brings a lot of new opportunities, it brings challenges too. There is a plenty of examples of applications that includes collective, social, distributed and even virtual/augmented reality features out there and, it just a matter to take a look to a smartphone of an average user to see several of them in action. But, besides the success behind laptops and cell phones as the mobile devices, currently there are many other hardware architectures which has mobile capabilities as well and, in which the implementation of new and interactive applications have increased in recent years. Weareable technolgy and ARM architectures are good examples of this.

Thanks to hardware advances, both companies and users are pushing to provide and experience better applications on mobile devices but this still has big impact in its performance, particularly in battery conpsumption. Research groups have focused on optimizing the hardware of mobile devices, as well as their middleware increasing both the devices' uptime and the user satisfaction but the greater demand in performance for both mobile device hardware and its applications conflicts with the desire for longer battery life.

Users are craving for longer battery life, some even claim that the next big thing in technology ought be better batteries [8]. Taking as example mobile applications, a study conducted by [21] analyzed 9 millions user comments and 27,000 mobile applications under the Google Play store and demostrated that users of mobile applications are interested in enery-efficient applications and energy-inefficient applications lead to frustrated users to negative feedback and lower application ratings. Some other relevants results of the study revealed that: more than 18% of the analyzed applications have user feedback comprising comments on energy consumption problems, energy-efficiency issues negatively influence the grades users give for mobile apps in market places and that free apps do not have more energy consumption problems than payed apps.

The energy use of a typical laptop computer, smartphone and a tablet is dominated by the backlight and display, disks and CPU. They use a number of techniques to reduce the energy consumed by the disk and display, primarily by turning them off after a period of no use. Power consumed by the CPU is becoming more significant because of the type of the applications they run now: data-intensive, multimedia, games and collaborative applications that need to be in a constant sync with networks, sensors, and other devices. Dynamic voltage scaling (DVS) is a common mechanism to save CPU energy. It exploits an important characteristic or CMOS[1]-based processors: the maximun frequency scales almost linearly to the voltage, and the energy consumed per cycle is proportional to the square of the voltage. A lower frequency hence enables a lower voltage and yields a quadratic energy reduction [3]. The major goal of DVS is to reduce energy by as much as possible without degrading appplication performance. The efectiveness of DVS techniques is, therefore, dependent on the ability to predict application CPU demands – overestimating them can waste CPU and energy resources, while understimating them can degrade aplication performance [26].

A real-time operating system has a well-specified maximum time for each action that it performs to support applications with precise timing needs. Systems that can guarantee these

This document was proposed as part of the *Advanced Topics in Operative Systems* course at Instituto Tecnológico de Costa Rica. First Semester, 2017. Taught by Ph.D Francisco Torres-Rojas http://dl.acm.org/author_page.cfm?id=81100442656

Abstract and References submitted and revised on March 20th, 2017.
First draft submitted on May 8th, 2017.

[1]CMOS: Complementary metal–oxide–semiconductor

maximum times are called hard real-time systems. Those that meet these times most of the time are called soft real-time systems. Deploying an airbag in response to a sensor being actuated is a case where you would want a hard real-time system. Decoding video frames is an example of where a soft real-time system will suffice

In this paper, we present [a set of] [the following] real-time scheduling algorithms [A, B, C] that have been proposed to improve energy-saving for mobile devices. We discuss their main features

II. Real-Time Task Scheduling Algorithms

Several schemes of classification of real-time task scheduling algorithms exist. A scheme in [14] classifies the real-time task scheduling algorithms based on how the scheduling points (the points on time line at which the scheduler makes decisions regarding which task is to be run next) are defined. According to this classification scheme, the three main types of schedulers are: clock-driven, event-driven, and hybrid. The clock-driven schedulers are those in which the scheduling points are determined by the interrupts received from a clock. In the event-driven ones, the scheduling points are defined by certain events which precludes clock interrupts. The hybrid ones use both clock interrupts as well as event occurrences to define their scheduling points.

Important examples of event-driven schedulers are Earliest Deadline First (EDF) and Rate Monotonic analysis (RM). Event-driven schedulers are more sophisticated and usually more proficient and flexible than clock-driven schedulers. These are more proficient because they can feasibly schedule some task sets which clock-driven schedulers can not. These are more flexible because they can feasibly schedule sporadic and aperiodic tasks in addition to periodic tasks, whereas clock-driven schedulers can satisfactorily handle only periodic tasks [14]. Event-driven scheduling of real-time tasks was a subject of intense research during early 1970s, leading to the publication of a large number of research results, of which the following two popular algorithms are the essence of all EDF and RM [12].

A. Earliest Dealine First (EDF)

Every process tells the operating system scheduler its absolute time deadline. The scheduling algorithm simply allows the process that is in the greatest danger of missing its deadline to run first. Generally, this means that one process will run to completion if it has an earlier deadline than another. The only time a process would be preempted would be when a new process with an even shorter deadline becomes ready to run. To determine whether all the scheduled processes are capable of having their deadlines met, the following condition must hold:

$$\sum_{i=1}^{n} \frac{C_i}{P_1} \leq 1$$

This simply tells us sum of all the percentages of CPU time used per process has to be less than or equal to 100%.

B. Rate Monotonic (RM)

Rate monotonic analysis is a technique for assigning static priorities to periodic processes. As such, it is not a scheduler but a mechanism for governing the behavior of a preemptive priority scheduler. A conventional priority scheduler is used with this system, where the highest priority ready process will always get scheduled, preempting any lower priority processes.

A scheduler that is aware of rate monotonic scheduling would be provided with process timing parameters (period of execution) when the process is created and compute a suitable priority for the process. Most schedulers that support priority scheduling (e.g., Windows, Linux, Solaris, FreeBSD, NetBSD) do not perform rate monotonic analysis but only allow fixed priorities, so it is up to the user to assign proper priority levels for all real-time processes on the system. To do this properly, the user must be aware of all the real-time processes that will be running at any given time and each process' frequency of execution ($1/T$, where T is the period). To determine whether all scheduled processes can have their real-time demands met, the system has to also know each process' compute needs per period (C) and check that the following condition holds:

$$\sum_{i=1}^{n} \frac{C_i}{P_1} < \ln 2$$

To assign a rate monotonic priority, one simply uses the frequency information for each process. If a process is an aperiodic process, the worst-case (fastest) frequency should be used. The highest frequency (smallest period) process gets the highest priority and successively lower frequency processes get lower priorities.

Scheduling is performed by a simple priority scheduler. At each quantum, the highest priority ready process gets to run. Processes at the same priority level run round-robin.

III. DVS Algorithms

There are two kinds of voltage scheduling approaches for hard real-time systems depdending on the voltage scaling granularity: intra-task DVS (IntraDVS) and inter-task DVS (InterDVS). The intra-task DVS algorithms adjust the voltage within an individual task boundary, while the inter-task DVS algorithms determine the voltage on a task-by-task basis at each scheduling point. The main difference between them is whether the slack times are used for the current task or for the tasks that follow. InterDVS algorithms distribute the slack times from the current task for the following tasks, while IntraDVS algorithms use the slack times from the current task for the current task itself.

A. Intra-task DVS algorithms

In scheduling hard real-time tasks, in order to guarantee the timing constraint of each task, the execution times of tasks are usually assumed to be the worst case execution times (WCETs). However, since a task has many possible execution paths, there are large execution time variations among them. So, when the execution path taken at run time is not the worst case execution path (WCEP), the task may complete its execution before its WCET, resulting in a slack time. In that case, IntraDVS exploits such slack times and adjusts the processor speed. IntraDVS algorithms can be classified into two types depending on how to estimate slack times and how to adjust speeds.

1) Path-based method: the voltage and clock speed are determined based on a predicted reference execution path, such as WCEP. For example, when the actual execution deviates from the predicted reference execution path, the clock speed is adjusted. If the new path takes significantly longer to complete its execution than the reference path, the clock speed is raised to meet the deadline constraint. On the other hand, if the new path can finish its execution earlier than the reference path, the clock speed is lowered to reduce the energy consumption. Program locations for possible speed scaling are identified using static program analysis [16] or execution time profiling [11].

2) Stochastic method: based on the idea that it is better to start the execution at a low speed and accelerate the execution later when needed than to start with a high speed and reduce the speed later when slack time is found. By starting at a low speed, if the task finishes earlier than its WCET, it does not need to execute at a high speed. Theoretically, if the probability density function of execution times of a task is known *a priori*, the optimal speed schedule can be computed [5]. Under the stochastic method, the clock speed is raised at specific time instances, regardless of the execution paths taken. The stochastic IntraDVS may not utilize all the potential slack times.

B. Inter-task DVS algorithms

InterDVS algorithms exploit the “run-calculate-assign-run” strategy to determine the supply voltage, which can be summarized as follows: (1) run a current task, (2) when the task is completed, calculate the maximum allowable execution time for the next task, (3) assign the supply voltage for the next task, and (4) run the next task. Most InterDVS algorithms differ during step (2) in computing the maximum allowed time for the next task τ which is the sum of WCET of τ and the slack time available for τ.

A generic InterDVS algorithm consists of two parts: slack estimation and slack distribution. The goal of the slack estimation part is to identify as much slack times as possible while the goal of the slack distribution part is to distribute the resulting slack times so that the resulting speed schedule is as uniform as possible. Slack times generally come from two sources; *static slack times* are the extra times available for the next task that can be identified statically, while dynamic slack times are caused from run-time variations of the task executions.

1) Slack estimation methods: Static slack estimation Maximum constant speed: One of the most commonly used static slack estimation methods is to compute the maximum constant speed, which is defined as the lowest possible clock speed that guarantees the feasible schedule of a task set [18]. For example, in EDF scheduling, if the worst case processor utilization (WCPU) U of a given task set is lower than 1.0 under the maximum speed f_{max}, the task set can be scheduled with a new maximum speed $f'_{max} = U \cdot f_{max}$. Although more complicated, the maximum constant speed can be statically calculated as well for RM scheduling [18, 5].

a) Dynamic slack estimation: Three widely-used techniques of estimating dynamic slack times are briefly described below.

Stretching to NTA: even though a given task set is scheduled with the maximum constant speed, since the actual execution times of tasks are usually much less than their WCETs, the tasks usually have dynamic slack times. One simple method to estimate the dynamic slack time is to use the arrival time of the next task [18]. (The arrival time of the next task is denoted by NTA.) Assume that the current task τ is scheduled at time t. If NTA of τ is later than $(t+\text{WCET}(\tau))$, task τ can be executed at a lower speed so that its execution completes exactly at the NTA.

Priority-based slack stealing: this method exploits the basic properties of priority-driven scheduling such as RM and EDF. The basic idea is that when a higher-priority task completes its execution earlier than its WCET, the following lower-priority tasks can use the slack time from the completed higher-priority task. It is also possible for a higher-priority task to utilize the slack times from completed lower-priority tasks. However, the latter type of slack stealing is computationally expensive to implement precisely. Therefore, the existing algorithms are based on heuristics [2, 10].

Utilization updating: the actual processor utilization during run time is usually lower than the worst case processor utilization. The utilization updating technique estimates the required processor performance at the current scheduling point by recalculating the expected worst case processor utilization using the actual execution times of completed task instances [14]. When the processor utilization is updated, the clock speed can be adjusted accordingly. The main merit of this method is its simple implementation, since only the processor utilization of completed task instances have to be updated at each scheduling point.

b) Slack distribution methods: In distributing slack times, most InterDVS algorithms have adopted a greedy approach, where all the slack times are given to the next activated task. This approach is not an optimal solution, but the greedy approach is widely used because of its simplicity.

IV. Representative Algorithms

Nine InterDVS algorithms are chosen, two of which are based on the RM scheduling policy, while the other seven algorithms are based on the EDF scheduling policy. Two algorithms were selected for IntraDVS, one from path-based IntraDVS algorithms, and the other from stochastic methods.

- `lppsEDF` and `lppsRM` which were proposed in [24], slack time of a task is estimated using the maximum constant speed and Stretching-to-NTA methods.
- `ccRM` algorithm proposed [15] is similar to `lppsRM` in the sense that it uses both the maximum constant speed and the Stretching-to-NTA methods. However, while `lppsRM` can adjust the voltage and clock speed

only when a single task is active, `ccRM` extends the stretching to NTA method to the case where multiple tasks are active.

- In [15], two other DVS algorithms are proposed, `ccEDF` and `laEDF`, for EDF scheduling policy. These algorithms estimate slack time of a task using the utilization updating method. While `ccEDF` adjusts the voltage and clock speed based on run-time variation in processor utilization alone, `laEDF` takes a more aggressive approach by estimating the amount of work required to be completed before NTA.
- `DRA` proposed in [2], is a representative DVS algorithm that are based on the priority-based slack stealing method. The algorithm estimates the slack time of a task using the priority-based slack stealing method along with the maximum constant speed and the Stretching-to-NTA methods.
- `AGR`, an extension of `DRA` proposed in [2] is intended for more aggressive slack estimation and voltage/clock scaling. In `AGR`, in addition to the priority-based slack stealing, more slack times are identified by computing the amount of work required to be completed before NTA.
- `lpSEH` in [9] is based on the priority-based slack stealing method. It extends the priority-based slack stealing method by adding a procedure that estimates the slack time from lower-priority tasks that were completed earlier than expected. `DRA`, `AGR`, and `lpSEH` algorithms are somewhat similar to one another in the sense that all of them use the maximum constant speed in the off-line phase and the Stretching-to-NTA method in the on-line phase in addition to the priority-based slack stealing method.
- `darEDF` in [27] is based on an efficient slack task scheduling scheme that employs dynamic speed setting slack in run queue. The algorithm uses a dynamic version of the average rate heuristic first and delays slack absorption, resulting in better battery performance.

Shin's intra-task DVS algorithm [23] and Gruian's algorithm [5] are used as representative IntraDVS algorithms which use path-based method and the stochastic method, respectively.

V. Results

In [10] a performance comparison of most of the algorithms presented in section IV was made using a simulation environment named SimDVS. It is argued that the energy efficiency of InterDVS algorithms depends significantly of slack estimation and the appropriateness of slack distribution. The following highlights the results obtained when evaluating InterDVS algorithms.

a) Impact of the number of task on the energy consumption: as the number of tasks increases, the energy efficiency of `lppsEDF`, `lppsRM`, and `ccRM` that only use the Stretching-to-NTA technique do not significantly improve, while that of the other more aggressive InterDVS algorithms improves significantly when compared with the energy consumption of an applicacion running on a DVS-unaware system with a power-down mode only. In the Stretching-to-NTA method, the slack time that can be exploited is limited to the time between the completion of a task instance and the arrival time of the next task instance, which is largely independent of the number of tasks in the system. For the other InterDVS algorithms, since the slack times can be taken from any completed task instance, as the number of task increases, each task has more slack sources and can be scheduled with a lowered clock speed.

b) Worst case processor utilization of task set: except for `lppsEDF`, the energy consumption of InterDVS algorithms increases as a linear function of WCPU of a task set. For `lppsEDF`, the energy consumption increases faster than a linear function of WCPU of a task set. This indirectly indicates that the dynamic slack estimation method of `lppsEDF` is not very effective. `lppsEDF` shows better energy efficiency than `ccEDF` when WCPU is less than 0.7. In `ccEDF`, the clock speed is determined using the actual processor utilization at the scheduling point. Since the actual processor utilization increases when a low-speed task instance completes its execution, the next task instance needs to be executed in a higher speed. Such voltage fluctuation occurs more often as the WCPU decreases. Thus, as the WCPU decreases, the energy efficiency of ccEDF becomes worse than that of `lppsEDF`. The results for `lppsRM` and `ccRM` are very similar to that of `lppsEDF`.

c) Speed bound: the energy efficiency of `AGR` and `lpSEH` is very close to the theoretical lower bound[2] when the speed bound factor is near 0.5. For the aggressive InterDVS algorithms, the energy efficiency is highest when the speed bound factor was set to average case processor utilization (ACPU). When the selected speed bound factor is close to ACPU ($= 0,55 \times$ WCPU), the best energy effciency is achieved for `laEDF`. For `ccEDF` this trend does not hold. For RM InterDVS algorithms, the performance gap between the energy efficiency and that of the theoretical lower bound was roughly 35∼40%.

==> Aqui Poner resultados de `darEDF` <==

A. Performance evaluation of Intra-Task DVS algorithms

The two representative IntraDVS algorithms perform quite differently depending on available slack times. When the relative energy consumption ratio of intraGruian over intraShin was measured, if the ratio is larger than 1, intraGruian performs better than intraShin. When the slack ratio is less than 1.2, intraShin outperforms intraGruian because intraShin spends more time in the lower speed region than intraGruian. When the slack ratio is increased, intraGruian spends more time in the lower speed region than intraShin.

VI. Related Work

- otros algoritmos para hard-time (lfpsEDF por ejemplo)
- scheduling en mobile devices basado en offloading

VII. Conclusions

References

[1] N. Audsley, A. Burns, R. Davis, K. Tindell and A. Wellings, *Real-time system scheduling*, 1995 Predictably Dependable Computing Systems, Springer Berlin Heidelberg, pp. 41-52.

[2] Determined by the Yao's algorithm in [25]

[2] H. Aydin, R. Melhem, D. Mosse, and P.M. Alvarez. *Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems*. In Proceedings of IEEE Real-Time Systems Symposium, December 2001.

[3] A. Chandrakasan, S. Sheng, and R.W. Brodesen. *Low-power CMOS digital design*. IEEE Journal of Solid-State Circuits, 27:473-484, April. 1992.

[4] R. Davis and A. Burns, *A survey of hard real-time scheduling for multiprocessor systems*, 2011 ACM computing surveys (CSUR), 43(4), 35.

[5] F. Gruian. *Hard Real-Time Scheduling Using Stochastic Data and DVS Processors*. In Proceedings of the International Symposium on Low Power Electronics and Design , pages 46–51, August 2001.

[6] N. Guan, W. Yi, Z. Gu, Q. Deng and G. Yu, *New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms*, 2008 Real-Time Systems Symposium, 2008. pp. 137-146. IEEE.

[7] C. Hung, J. Chen and T. Kuo, *Energy-Efficient Real-Time Task Scheduling for a DVS System with a Non-DVS Processing Element*, Real-Time Systems Symposium, 27th IEEE International, Rio de Janeiro, Brazil, 2006. doi: 10.1109/RTSS.2006.22

[8] D.Moren. *The next big thing in tech ought to be better batteries*. Macworld. August 19, 2016. Available at http://www.macworld.com/article/3109692/hardware/the-next-big-thing-in-tech-ought-to-be-better-batteries.html

[9] W. Kim, J. Kim, and S.L. Min. *A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis*. In Proceedings of Design, Automation and Test in Europe (DATE'02), pages 788–794, March 2002.

[10] W. Kim, J. Kim, and S.L. Min *Performance comparison of dynamic voltage scaling algorithms for hard real-time systems*. proc. RTAS. pp 219-228. 2002.

[11] Y. W. Kwon and E. Tilevich, *Reducing the Energy Consumption of Mobile Applications Behind the Scenes*, 2013 IEEE International Conference on Software Maintenance, Eindhoven, 2013, pp. 170-179. doi: 10.1109/ICSM.2013.28

[12] W.S. Liu. *Real-Time Systems*. Prentice Hall, Englewood Cliffs, NJ, June 2000.

[13] J. Luo and N.K. Jha, *Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems*, IEEE/ACM International Conference on Computer Aided Design, San Jose, CA, 2000. doi: 10.1109/ICCAD.2000.896498

[14] R. Mall. *Real-Time Systems: Theory and practice*. Pearson Education India, 2009.

[15] P. Pillai and K.G. Shin. *Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems*. In Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01), pages 89–102, October 2001.

[16] H. Qian and D. Andresen, *An energy-saving task scheduler for mobile devices*. 2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS), Las Vegas, NV, 2015, pp. 423-430. doi: 10.1109/ICIS.2015.7166631

[17] V. Rao, N. Navet and G. Singhal, *Battery aware dynamic scheduling for periodic task graphs*. 2006 Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, Rhodes Island, 2006. doi: 10.1109/IPDPS.2006.1639403

[18] H. Takada and K. Sakamura, *Real-time synchronization protocols with abortable critical sections*. 1994 Proceedings of 1st International Workshop on Real-time Computing Systems & Application, pp. 48-52

[19] C. Tianzhou, H. Jiangwei, X. Lingxiang and W. Xinliang, *Balance the battery life and real-time issues for portable real-time embedded system by applying DVS with battery model*, 34th Annual Conference of IEEE, Florida Hotel & Conference Center, FL, 2008. doi: 10.1109/IECON.2008.4758018

[20] M. Weiser, B. Welch, A. Demers, and S. Shenker. *Scheduling for reduced CPU energy*. In Proc. of Symposium on Operating Systems Design and Implementation. Nov. 1994.

[21] C. Wilke, S. Richly, S. Götz, C. Piechnick and U. Aßmann, *Energy Consumption and Efficiency in Mobile Applications: A User Feedback Study*, 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, Beijing, 2013, pp. 134-141. doi: 10.1109/GreenCom-iThings-CPSCom.2013.45

[22] R. Ravishankar, V. Sarma and R. Daler N, *Battery modeling for energy aware system design*, Computer, IEEE Xplore Digital Library, Volume 36, Pages 77-87, 2003

[23] D. Shin, J. Kim, and S. Lee. *Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications*. IEEE Design and Test of Computers, 18(2):20–30, March 2001

[24] Y. Shin, K. Choi, and T. Sakurai. *Power Optimization of Real-Time Embedded Systems on Variable Speed Processors*. In Proceedings of the International Conference on Computer-Aided Design, pages 365–368, November 2000.

[25] F. Yao, A. Demers, and A. Shenker. *A Scheduling Model for Reduced CPU Energy*. In Proceedings of the IEEE Foundations of Computer Science, pages 374–382, October 1995.

[26] W. Yuan and K. Nahrstedt, *Energy-efficient soft real-time CPU scheduling for mobile multimedia systems*, ACM SIGOPS Operating Systems Review, Volume 37, Pages 149-163, 2003.

[27] J. Zhuo and C. Chakrabarti, *An efficient dynamic task scheduling algorithm for battery powered DVS systems*, 2004 IEEE International Symposium on Circuits and Systems, Vancouver, BC, 2004. doi: 10.1109/ISCAS.2004.1329396