Response Time in Mixed-Critical Pervasive Systems

Sudharsan Vaidhun, Samsil Arefin, Zhishan Guo, Haoyi Xiong, Sajal K. Das
Department of Computer Science, Missouri University of Science and Technology
{svqm5,sazhx,guozh,xiongha,sdas}@mst.edu

Abstract

Pervasive computing systems at large scale rely on real-time scheduling on the top of distributed and networked computing environments. From an user experience perspective, while the requirements on the response time for specific applications might be different, the mixed-criticality in real-time scheduling, which provide diverse response time guarantee for applications, is often required. In this paper, we study the real-time scheduling problem in mixed-critical pervasive computing systems. We first analyze the response time requirements for common networked pervasive computing systems, and model the mixed-criticality using the minimum response time Quality-of-Service (QoS) that should be guaranteed even in the worst-case. Then, we propose to leverage Fixed-Priority Rate-Monotonic (FPRM) Scheduler for real-time scheduling. We evaluate FPRM using synthetic workloads generated according to the real-world pervasive computing systems. Both simulation experiments and worst-case analytical results show that, when sufficient resources are given, all pervasive computing tasks can be completed subject to the response time requirements strictly with mixed-criticality guarantees ensured.

1. Introduction

Pervasive computing, also known as ubiquitous computing, refers to integrating everyday objects with computational capability into a networked system to enable a seamless computing experience for the user. Such experience may refer to a wide variety of applications ranging from a simple peer-to-peer communication and data-intensive media streaming to highly interactive online gaming. One such application, the real-time entertainment (e.g. audio and video streaming) is growing at an immense rate which contributed to about 36% [1] of the mobile traffic in 2016. This signifies a growing demand for platforms that can handle real-time traffic.

Mixed-Criticality. When a system with multiple applications is considered, some of these applications are more *critical* than others. For example, a car navigation application is more critical than a real-time entertainment or radio environment in a car. Moreover, some are time-critical while some others are quality-critical. The quality measure of a task is given in terms of response time QoS. The criticality in a car navigation application is the timeliness of the output; a delayed information from the application is as good as no information at all. Whereas, in a media entertainment system, although there could be a lag in the stream, as long as it is within a tolerable range it is not noticeable by the user. Due to the size, weight, and power considerations, there is a trend in combining mixed-critical applications with varying specifications upon a shared platform [6] and such systems are called *Mixed-Critical systems (MCS)*.

When designing mixed-critical applications for a shared platform, to avoid contention for resources, the access to resource follows a schedule. So, when an application requires access, it must wait for its turn in the schedule. The *response time* of the task therefore depends on the schedule. As mentioned earlier, for a critical application, it is important to know the response time of the request, in order to ensure that it is scheduled before its deadline. One approach to analyzing the expected wait time and response time of the task is queuing theory. In queuing theory, the task arrival time and the processing time is assumed to follow a distribution and the analysis of these tasks for a given scheduling algorithm and a set of tasks gives us the *mean waiting time* and *mean response time* for the individual tasks.

Towards designing scheduling algorithms with criticality of tasks in consideration, Vestal [13] was the first to verify fixed priority scheduling for mixed criticality systems. Later, when soft tasks (as opposed to hard real-time tasks with hard deadlines) were considered. For soft real-time tasks, the worthiness of the output degrades with increasing tardiness (i.e. response time greater than deadline). Such tasks are not deadline-critical, but rather quality-critical. Initial techniques guaranteed hard tasks and used the remaining resource available to schedule other soft tasks. This approach did not have any bounded guarantee on the low critical task. Recent work by Hu et al. [9], [10] adaptively shaped the

978-1-5386-0435-9/17/$31.00 ©2017 IEEE

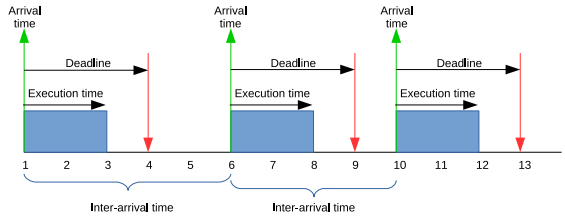

Figure 1. Sporadic Task.

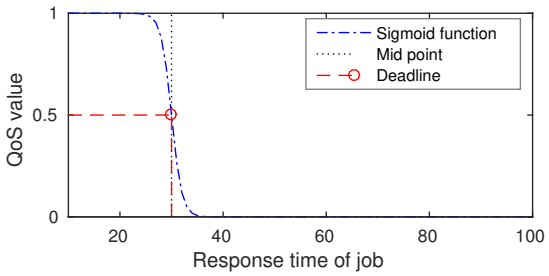

Figure 2. Deadline-Critical Task; $m_i = 30$; $k_i = 1$.

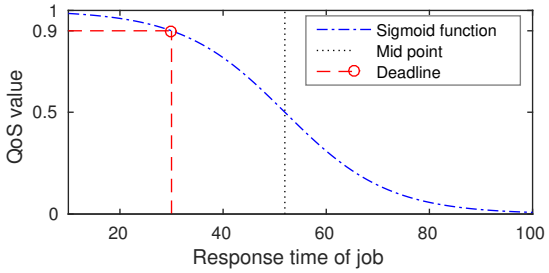

Figure 3. Quality-Critical Task; $m_i = 52$; $k_i = 0.1$.

low critical task workload so that the QoS of lower critical tasks can be improved.

However, despite some recent work under mixed-criticality with graceful degradation, correctness guarantees for soft real-time tasks are normally not provided [11], [3], [8]. In our work, we introduce a model to represent mixed-critical tasks using a QoS function. By adjusting the parameters of the QoS function, we denote the quality expected to be guaranteed to each task. The QoS function gives a quantitative value for each task in the system for a given scheduler. The QoS values of each task is evaluated for common scheduling algorithms.

The remainder of this paper is organized as follows. Section 2 formally defines the system model and QoS metric for mixed-critical tasks. Section 3 discusses the scheduling techniques used for analysis of QoS measure of the tasks. Further, Section 4 shows the simulation results and our inference from the experiments. The conclusions are drawn in the final section.

2. System Model

In this section, we formally introduce the models and metrics. We represent the requests generated by the different applications in a system as a set of job sequence. Formally, a task-set τ is a collection of tasks τ_i. Each task τ_i generates a sequence of jobs $\tau_{i,j}$ and has an associated criticality level. The criticality levels are modeled by the Quality-of-Service (QoS) function described in the following subsections.

2.1. Sporadic task model

We consider a general system, where nodes send requests to the server for processing, and the processing of the tasks takes a pre-defined finite amount of time. We model this using a *sporadic task model* [12], in which the task τ_i is represented by a tuple

$$\tau_i = \{c_i, d_i, t_i\} \tag{1}$$

where, c_i is the worst-case execution time of the jobs in the task, t_i is the minimum inter-arrival time between two instants of jobs from task τ_i, d_i is the deadline for the task. The deadline d_i is relative to the release time (or arrival time) of the job in the task. The ratio $u_i = \frac{c_i}{min(d_i,t_i)}$ is the utilization of a task τ_i. For a set of tasks $\{\tau_i \in \tau\}$, the total utilization is $U_i = \sum u_i$

Figure 1 shows a sequence of jobs of a sporadic task $\tau_i = \{2,3,3\}$. It can be seen that the execution time $c_i = 2$ and deadline $d_i = 3$ for all the jobs. It can also be seen that the minimum inter-arrival time between two jobs is greater than or equal to $t_i = 3$.

2.2. Mixed-Criticality

As seen earlier, each task has an associated criticality – deadline-critical or quality-critical. First, let us consider the case of a deadline-critical task. For a deadline-critical task, the meaningfulness (or quality) of the response quickly degrades as the response time of the job goes beyond its deadline. Second, for a quality-critical task, the quality of the response gradually degrades with increasing response time. By adjusting the gradient of the degradation, we can model varying levels of quality.

In summary, by adjusting the gradient of the degradation, we can effectively represent the expected quality of a task.

2.3. Quality-of-Service (QoS)

The mixed-criticality levels mentioned in the previous sub-section and the degradation in quality can be

quantified using the QoS function. Other works in the literature [7] [14] have shown that a sigmoid function can be used to adequately represent the QoS provided to the user.

We define the QoS of a single job $\tau_{i,j}$ as

$$QoS_{i,j} = \frac{1}{1 + e^{-k_i(r_{i,j} - m_i)}} \quad (2)$$

where, $r_{i,j}$ is the response time of the job $\tau_{i,j}$. And m_i and k_i are the midpoint and slope of the sigmoid function, which are design parameters chosen for the task.

The QoS of the task is represented in terms of the QoS of its jobs

$$QoS_i = \frac{1}{n} \sum_j QoS_{i,j}.P(r_i) \quad (3)$$

where, $P(r_i)$ represents the probability distribution of the response times of the task. Essentially, the QoS of a task is the average QoS of all the jobs generated by the task.

In Figures 2 and 3, we show the QoS sigmoid curve for a deadline-critical task and a quality-critical task. The QoS function parameters are chosen in such a way that it reflects the degradation in value as the response-time of the task approaches and crosses the deadline. For a deadline-critical task, this is achieved by setting the midpoint $m_i = 30$ which is the same as its deadline d_i and the slope k_i to a large value of 1.0. Assigning a vary large slope value is analogous to representing a hard real-time task. Figure 2 shows that the QoS begins to degrade at time $r_{i,j} = 25$ and proceeds to drop rapidly to 0 when the time reaches $r_{i,j} = 35$. For even higher slope values, the QoS will drop almost immediately as it goes beyond the deadline.

For a quality-critical task this is achieved by taking the midpoint $m_i = 52$ which is larger than its deadline $d_i = 30$ and a lower slope value ($k_i = 0.1$). Figure 3 shows the QoS slowly degrading as time proceeds and it reaches close to 0 at around $r_{i,j} = 100$ time units.

3. Scheduling and Analysis

After defining our task model and the QoS metrics associated with the different criticalities of the tasks, we use existing simple scheduling algorithms to schedule our sample task-set and analyze the QoS for each scheduling technique. We first consider a simple Round-Robin (RR) scheduling algorithm and the real-time scheduling algorithms like Fixed-Priority Rate-Monotonic (FPRM) scheduler. The algorithms are briefly explained below followed by QoS analysis.

3.1. Round-Robin (RR) scheduling algorithm

The basic RR is a dynamic scheduling algorithm that schedules the jobs in a circular order. Each released job is time sliced into *time quanta* and a slice of every job is scheduled one-by-one. More advanced versions of RR consider the priority level of the tasks while scheduling. Our application of RR algorithm does not consider the priority or the criticality of the task. We schedule our mixed-critical task-set using RR algorithm and measure the QoS of the tasks after scheduling.

3.2. Fixed priority scheduling algorithms

Beyond simple algorithms like RR, in real-time scheduling theory, it is necessary to know apriori whether the scheduler can guarantee the correctness of the task scheduled. There are two broad approaches for task scheduling in real-time systems – static scheduling (clock-driven) and dynamic scheduling (event-driven or priority-driven). Since our task load is non-deterministic, we cannot use static scheduling approaches. For further discussion, we will consider priority driven scheduling algorithms in which the priority may be static or dynamic. A static priority scheduling algorithm assigns the same priority to all jobs of a task while for dynamic priority algorithms, the priority is assigned to every job at its release time. The scheduling techniques considered in our work are discussed below.

- In the first approach, FPRM scheduling is applied without considering the criticality of the task. The tasks are assigned priority based on their period t_i. Tasks with smaller periods get higher priority. When a higher priority job arrives, all lower priority jobs are stopped from executing and the high priority job is scheduled. Since we did not consider the criticality of the job, we will later show experimentally that the QoS value of the deadline-critical jobs degrades as the system utilization increases.

- In the second approach, criticality-first FPRM, we modify the FPRM scheduling to improve the QoS of deadline-critical tasks. Priorities are assigned to deadline-critical tasks first using FPRM and later to QoS-critical tasks. Since the deadline-critical jobs are assigned higher priority, this scheduler is biased towards scheduling deadline-critical jobs.

Other Scheduling algorithms. Dynamic priority scheduling algorithms such as Longest-Execution-Time-First (LETF) and Shortest-Execution-Time-First (SETF) algorithms assign priority based on the execution time of the jobs. The scheduling algorithms used

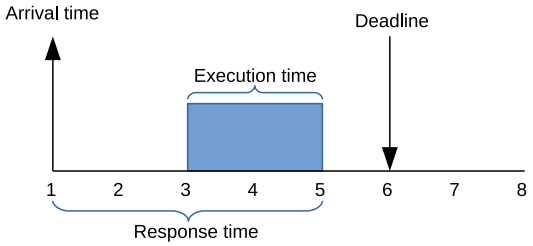

Figure 4. Response time of a job $\tau_{i,j}$.

in non real-time applications such as First-In-First-Out (FIFO) and Last-In-First-Out (LIFO) assign priority to the job based on their release time.

3.3. Response Time and QoS

The response-time of a job is defined as the difference between its completion time and release (or arrival) time. For example, consider Figure 4. A job with worst-case execution time $c_i = 2$ is released at time 1 and its scheduled for processing at time 3, then the response-time of the job is $r_i = 3+2-1 = 4$. The complexity of calculating this response time for entire task-set has a polynomial time complexity.

3.3.1. Response time bound. To reduce the computational complexity of calculating the response time, Bini et. al proposed a method [5] to calculate the worst case response time (upper bound) of sporadic tasks with arbitrary deadline as

$$r_i \leq \frac{c_i + \sum_{j<i} c_j(1-u_j)}{1-\sum_{j<i} u_j} = R_i^{ub} \tag{4}$$

Here, the R_i^{ub} is the upper bound on the response time for task τ_i. Using Eq. (4) to calculate the worst-case response time has linear time complexity. This technique reduces the computational complexity at the cost of pessimistic response time bound.

3.3.2. Expected response time. The response time bound calculated above gives us the worst case QoS. To calculate the average case QoS, we use queuing models to compute the expected response time of the tasks. We consider the $M/M/1$ queuing model [2] to analyze our task-set. The arrival time is assumed to follow exponential distribution and the service time is taken as the execution time c_i of the task. If the mean inter-arrival time is $1/\lambda$ and the mean service time is $1/\mu$, then the system utilization U is $\rho = \lambda/\mu$ where $\rho < 1$ for a feasible system. If the arrival rate is more than the service rate, the system cannot feasibly schedule all the tasks before their deadline. Under these conditions, the distribution function of response time is given by

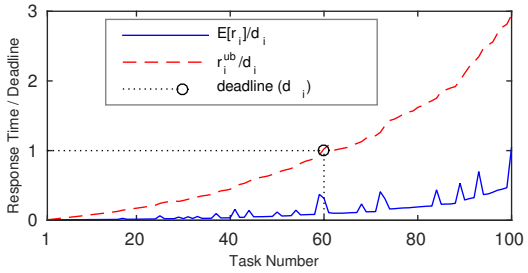

Figure 5. Comparison of $\frac{R_i^{ub}}{d_i}$ and $\frac{E[r_i]}{d_i}$ for a task-set with 100 sporadic tasks and system utilization $U = 0.75$.

$$P(t) = P(r_i \leq t) = 1 - e^{-\mu(1-\rho)t} \tag{5}$$

and the mean response time is given by

$$E[r_i] = \frac{1}{\mu(1-\rho)} \tag{6}$$

The parameters of the exponential distribution for arrival time are chosen to approximate the distribution of release time of the tasks. Figure 5 plots the ratio of expected response time and the deadline of the tasks and compare it with the ratio of the worst-case response time and the deadline of the task in Figure 5. It can be seen that the upper bound on the response time upper bound envelopes the expected response time of the tasks. For our task-set of 100 sporadic tasks, 60 tasks are guaranteed to be scheduled before the deadline and the remaining 40 tasks experience increasing delay in the response time. It should be noted that the distribution is only an expected response time and does not consider the actual experimental result.

3.3.3. QoS calculation. The QoS of the task shown in Eq. (2) is used to calculate the quality guaranteed to each task for the response times. We calculate the worst-case QoS using the response time upper bound r_i^{ub} values, the expected QoS using the expected response time $E[r_i]$ and the actual run-time QoS via simulation. For calculating the actual QoS, we generate a task-set with two types of tasks – deadline-critical and quality-critical. We schedule the generated task-set using RR and FPRM scheduling algorithms and calculate the QoS for each job and further for each task. The experimental results are shown in the next section.

4. Simulation Results

In this section, we validate our QoS function for representing the mixed-criticality of the tasks via simulation experiments. First we calculate the expected

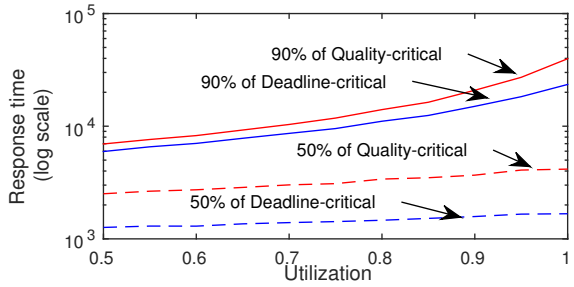

Figure 6. Response time of tasks using FPRM.

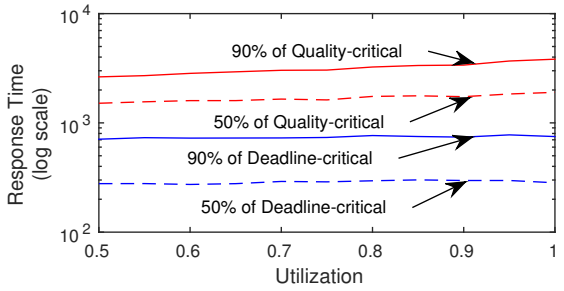

Figure 7. Response time of tasks sorted by criticality and scheduled using FPRM.

and worst-case response bounds for the tasks without scheduling them. Later we schedule them and calculate the actual response time and the QoS of the tasks.

4.1. Experiment 1: Response time Calculation

We consider a task-set with 150 tasks. The overall system utilization is $U = \sum_i u_i$ such that $U \leq 1$ where $u_i = \frac{c_i}{min(d_i,t_i)}$. We perform the simulation for utilization values u_i ranging from 0.5 to 1.0 in increments of 0.05. The period of the tasks in the system ranges from 10^3 to 10^6 time units following lognormal distribution. The system utilization is distributed among the tasks using UUniFast algorithm [4]. For a desired number of tasks and system utilization, the UUniFast algorithm generates a task-set with system utilization randomly distributed among the tasks. The execution time of the task is based on the utilization of the task and the period of the task.

We implement only the FPRM scheduling techniques to compare the expected response time and worst-case response time (upper bound). First, we do not consider the criticality of the tasks and implement fixed priority rate-monotonic scheduling as discussed in section 3. Second, we repeat the same experiment sorting tasks by their criticality and then applying rate-monotonic scheduling.

For analysis of the worst-case response time we calculate the time taken to complete 50% and 90% of

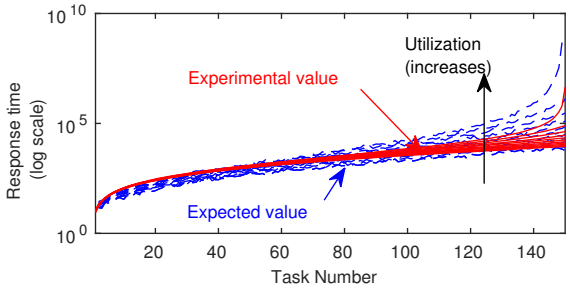

Figure 8. Comparison of experimental mean response time and analytical expected response time for FPRM.

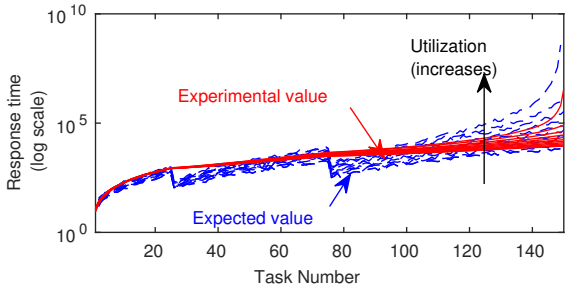

Figure 9. Comparison of experimental mean response time and analytical expected response time for criticality-first FPRM.

the tasks for both the scheduling algorithms. This simulation result is presented in Figures 6 and 7. From Figure 6, we observe that while 50% tasks of all priorities have lower response time at fully loaded condition ($U = 1$), we can also see that the high priority tasks have significantly high response time. This is undesirable because guaranteeing a low response time for high priority tasks is necessary. Figure 7 shows that 90% of the high priority tasks have significantly lower response time, but the downside is that 50% of the low priority tasks take significantly longer time to respond. This could provide a very low quality of service for the low priority applications.

Further, for varying utilization values, we plot the expected mean response time of the tasks and the worst-case response times derived using Eq. (6) for both the algorithms and are shown in Figures 8 and 9. These figures show the comparison between the experimental average response time of the tasks for different utilization values and the analytical estimation of the response time. When the system load increases, the queuing model tends to overestimate the response time. Figure 9 shows sharp bumps in the response time between different criticality classes, but the experimental average case is rather more smooth.

Table 1. Average QoS of Mixed-critical task-set

Util.	Rate-Monotonic		Rate-Monotonic w Criticality		Round-Robin	
	High	Low	High	Low	High	Low
0.5	1.0	1.0	1.0	1.0	1.0	1.0
0.6	1.0	1.0	1.0	1.0	1.0	1.0
0.7	1.0	0.999	1.0	0.999	1.0	0.999
0.8	1.0	0.998	1.0	0.992	0.999	0.986
0.9	1.0	0.877	1.0	0.958	0.960	0.932
1.0	1.0	0.815	1.0	0.737	0.813	0.837

4.2. Experiment 2: QoS Calculation

We consider an implicit deadline task-set of 150 tasks with periods arbitrarily chosen in the range $(100, 500)$. The tasks are divided into two critical levels with QoS function slope parameters k_i of 1 and 0.1 for high and low critical tasks, respectively. The task-set is scheduled for all three scheduling algorithms for varying utilization rates. The system is overloaded beyond $U = 1$ and the average QoS of the tasks of each criticality level is calculated.

Table 1 shows the average QoS of the tasks of different criticality for different scheduling algorithms. It can be observed that the FPRM scheduling algorithm which does not consider the criticality, balances the load between high and low criticality tasks at overloaded conditions. But, the criticality-first FPRM guarantees the high critical tasks and then guarantees lower critical tasks depending on the resource availability.

5. Conclusion

The use of real-time scheduling theory task models for representing mixed-critical pervasive systems aids us in quantifying the *performance* as well as the *predictability* of the system in terms of *response time* experienced by the user. Adding to this, a QoS value for each such task of all criticality levels quantifies the quality guaranteed for each task. Such modeling gives the service provider or the designer an insight on the worst-case quality of the system. Given this knowledge, it is easy to design admission control mechanisms for the users in the system. The admission control procedures can also be explicitly designed to guarantee a required level of satisfaction for a given set of users/nodes, system utilization and criticality of the tasks.

Following the initial success with common scheduling algorithms, the QoS values for state-of-the-art mixed-critical algorithms needs to be analysed. Future work also includes, designing schedulers that take QoS into consideration while scheduling jobs.

References

[1] Sandvines Internet Phenomena Blog : Global Internet Phenomena Report 2016: Latin America & North America.

[2] I. Adan and J. Resing. *Queueing Systems*. Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands, March 2015.

[3] S. Baruah, A. Burns, and Z. Guo. Scheduling Mixed-Criticality Systems to Guarantee Some Service under All Non-erroneous Behaviors. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 131–138, July 2016.

[4] E. Bini and G. C. Buttazzo. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, 30(1-2):129–154, May 2005.

[5] E. Bini, T. H. C. Nguyen, P. Richard, and S. K. Baruah. A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines. *IEEE Transactions on Computers*, 58(2):279–286, February 2009.

[6] A. Burns and R. I. Davis. Mixed criticality systems - A review. *Technical Report MCC-1(H), available at http://www-users.cs.york.ac.uk/burns/review.pdf*, 2016.

[7] M. Chatterjee, H. Lin, and S. K. Das. Rate Allocation and Admission Control for Differentiated Services in CDMA Data Networks. *IEEE Transactions on Mobile Computing*, 6(2):179–191, February 2007.

[8] Z. Guo, K. Yang, S. Arefin, S. Vaidhun, and H. Xiong. Uniprocessor Mixed-Criticality Scheduling with Graceful Degradation. In submission.

[9] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. Adaptive Runtime Shaping for Mixed-criticality Systems. In *Proceedings of the 12th International Conference on Embedded Software*, EMSOFT '15, pages 11–20, Piscataway, NJ, USA, 2015. IEEE Press.

[10] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. Adaptive Workload Management in Mixed-Criticality Systems. *ACM Trans. Embed. Comput. Syst.*, 16(1):14:1–14:27, October 2016.

[11] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi. EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 35–46, November 2016.

[12] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. Thesis (Ph. D.), Massachusetts Institute of Technology, 1983.

[13] S. Vestal. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 239–243, December 2007.

[14] M. Xiao, N. B. Shroff, and E. K. P. Chong. Utility-based power control in cellular wireless systems. In *Proceedings IEEE INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society*, volume 1, pages 412–421, 2001.