Online Workload Monitoring with the Feedback of Actual Execution Time for Real-Time Systems

Biao Hu[1],Kai Huang[2], Gang Chen[3], Long Cheng[1], Alois Knoll[1]
[1]Tech. Univ. Muenchen TUM, [2]Sun Yat-Sen University, [3]Northeastern University
[1]{hub,chengl,knoll}@in.tum.de, [2]huangk36@mail.sysu.edu.cn, [3]chengang@cse.neu.edu.cn

Abstract— Guaranteeing the system workload within design bounds is a basic requirement for a real-time system. Design-time bounds are usually based on worst-case activation patterns and worst-case execution time. While using the worst-case assumptions for online monitoring can guarantee the system safety, it also introduces unexplored slacks due to tasks consuming less than their worst-case execution times. In this paper, we introduce a monitoring scheme with the feedback of actual execution time for real-time systems. By using this runtime feedback instead of offline assumptions, this monitoring scheme can accept events that are considered as violations offline, and thereby improve the system utilization. In the experiments of both MATLAB simulation and MicroC/OS-II running in a softcore processor implemented on an FPGA, different probability distributions of actual execution time are used in analyzing how much the benefit can be gained from the feedback scheme.

I. INTRODUCTION

In the analysis of real-time systems, designers often employ worst-case assumptions to verify the system adherence to timing constraints. The worst-case assumptions involve the task activations and the task execution time. A common model of task activations is the *arrival curve* which provides bounds on the number of activations that can be accepted in any time interval [3], [17]. The worst-case task activations are activations that comply with the arrival curve. The worst-case execution time (WCET) of a computational task is the maximum time length that the task could take to execute on a specific hardware platform. Another common model that combines the task activations and the execution time is the *workload arrival function* (WAF) [19]. The WAF specifies the worst-case processing cycles that can be requested by a task or a group of tasks in any time interval.

During the runtime, model-based monitors are often used to verify the correctness of system timing properties. Event-based monitors verify that task activations do not exceed the designed upper bound by employing arrival curve [8], [10], [14], [15]. Workload-based monitors verify that the system workload does not exceed the designed upper bound by employing WAF [13]. The designed upper bound of either events or workload is based on the assumption that a task needs WCET execution time to sufficient guarantee its timing requirement.

While the WCET assumption in the design-time analysis can guarantee the system safety (all task deadlines can be met) in the worst cases, the runtime verification with the assumption of the WCET is somehow pessimistic. In most cases, the actual execution time (AET) of a task is a time length between BCET (best-case execution time: the minimum execution time in a specific hardware platform) and WCET, which indicates that there are always some slacks unexplored between the actual and the assumed worst-case workload. An example is shown in Fig. 1, where the worst-case workload is the workload that adopts the WCET as its AET; runtime workload is the measured workload that considers the AET of every unfinished event as the WCET and reduce the difference between the AET and

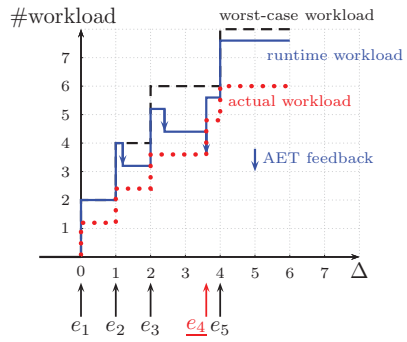


Fig. 1: The monitoring impact of worst-case and actual workload during the runtime

the WCET after an event is finished; the actual workload is the workload that accumulates all the AETs. Note that we assume the actual workload is clairvoyant to know the AET before an event is finished, which is not realistic during the runtime. In this example, the WCET of this task is 2, and the AET is 1.2. Events e_i denote task activations. If the WCET is used for verification during the runtime, event e_4 will be considered as a violated event. But actually, event e_4 is not a violated event because the actual workload (dotted curve) is still below the worst-case workload at e_4 (dashed curve). This example shows that the monitor is more tolerable if AET is used in the runtime monitoring. While it is straightforward that the monitor can accept more events by using AET to replace WCET during the runtime, how much the benefit can be gained still depends on how much the slack can be explored. Therefore, the distribution of AET in the interval [BCET, WCET] has a great influence on the effect of such AET feedback scheme.

In this paper, a runtime monitor with AET feedback is proposed in order to improve the system utilization. For the single event stream, the monitor is based on the so-called dynamic counters [8], [10], which itself is derived from the arrival curve model. By adopting the arrival curve to model the incoming system workload as event streams, our monitor can cope with non-deterministic arrival patterns of the incoming workload. The monitor makes use of the actual processing time of incoming events, i.e., the notification from the system for event finishes, and re-calculate the remaining processing capacity. In this way, slacks between the AET and the WCET can be used to update the runtime workload to help the monitor to verify the system workload more tightly. The monitoring performance is tested on MATLAB simulation and in MicroC/OS-II running in FPGA by applying different probability distributions of AET.

The remainder of this paper is structured as follows. Section II reviews related work. Section III introduces the system models that our monitoring is applied in. Section IV provides the detailed algorithm of applying the AET feedback scheme in monitoring. Section V prototypes the monitoring algorithm

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

as a hardware IP and compares the monitoring performance between with and without AET feedback of monitoring single event stream. Section VI concludes this paper.

II. Related work

Runtime monitoring is of great importance as a safe guard to guarantee the correctness of system runtime behaviors. Most monitoring approaches mainly focus on monitoring a specified system state during the runtime. The state can be program function [9], or correct combination of software and hardware [1]. The focus of this paper is that the runtime workload will not exceed our given upper-bound over any time interval. The task workload of a fixed-length interval consists of two components: the task activations within this interval and the task worst case execution time. The execution time is often counted by the timer. Common mechanisms to perform execution time monitoring are through watchdog timers - either in hardware or in software [2]. In this paper, we assume the execution time will not exceed the WCET, and the aim of this paper is to verify the task activations.

A common approach to model the task activations is the arrival curve (or the dual representation of minimum distance function). The concept of arrival curve in Real-Time Calculus theory establishes a link between network communication theory and real-time scheduling analysis [12], [17]. By assuming the WCET of task executions, a schedulable task activation pattern can be obtained. Different monitoring approaches are proposed for different task activation patterns.

With the observations that arrival curve can be conservatively approximated by a set of staircase functions, Lampka *et al.* [10] proposed using dynamic counters to predict the future coming events. This method is extended by Huang *et al.* [8] to use dual leaky-bucket counters to monitor the staircase function. The overhead is shown to be very low in FPGA implementation. Neukirchner *et al.* [14] presented a light-weight monitoring approach for arbitrary activation patterns. The task activation patterns are modelled as minimum distance functions, which describe lower bounds on the temporal distance between consecutive activations. To decrease the monitoring overhead, a l-repetitive minimum distance function is constructed to represent the minimum distance function. The comparisons between the dynamic counters monitoring and l-repetitive function and further improvements based on them are presented in [6], [7].

Except the single event stream monitoring, there are also some group tasks monitoring approaches. The authors in [15] proposed a scheme for monitoring activation patterns of multiple streams in mixed-criticality real-time systems. This scheme provides a methodology, which is based on the interface design [20] and sensitivity analysis [16], to derive sets of activation pattern bounds for each monitored software component. In this way, slacks between different tasks can be explored to schedule more task activations. Another monitoring method is the workload-based monitoring [13]. The system workload is a combination of the task activations and execution time [19]. Workload-based monitoring is able to express correlations between the activations of different tasks, and improves resource utilization as no isolation between low-critical tasks in a group is enforced. The system scheduling performance can be improved by relying on the online monitoring proposed in [4], [5].

All aforementioned monitoring methods assume the task is executed for the WCET. Although this assumption can guarantee the system safety, it also introduces some false negative detections (erroneously rejects non-violated events). By using the AET, instead of the WCET, our monitor can reduce the number of false negative detections while keeping the system safety, thereby improves the system utilization.

III. System model

In our system, we assume a task is activated by an activating event. The activating events can be expressed as an event stream. A trace of such an event stream can be conveniently described by means of a differential arrival function $R[s,t)$ that denotes the sum of events arrived in the time interval $[s,t)$, with $R[s,s) = 0, \forall s,t \in \mathbb{R}$. While any R always describes one concrete trace, a 2-tuple $\overline{\alpha}(\Delta) = [\overline{\alpha}^u(\Delta), \overline{\alpha}^l(\Delta)]$ of upper and lower arrival curves provides an abstract event stream model that characterizes a whole class of (non-deterministic) event streams. $\overline{\alpha}^u(\Delta)$ and $\overline{\alpha}^l(\Delta)$ provide an upper and lower bound on the number of events seen on an event stream in any time interval of length Δ [18]:

$$\overline{\alpha}^l(t-s) \leq R[s,t) \leq \overline{\alpha}^u(t-s), \forall t \geq s \geq 0$$

with $\overline{\alpha}^l(\Delta) = \overline{\alpha}^u(\Delta) = 0$ for $\Delta \leq 0$.

Analogous to arrival curves that provide an abstract event stream model, a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower service curves then provide an abstract resource model. The upper and lower service curve provides an upper and lower bound on the available resources in any time interval of length Δ. An arrival curve $\overline{\alpha}_i(\Delta)$ specifies the (upper-bounded or lower-bounded) number of events of event stream S_i for any time interval Δ, while a service curve $\beta(\Delta)$ specifies the (upper-bounded or lower-bounded) available amount of time for execution for any time interval Δ. Therefore, the arrival curve $\overline{\alpha}_i^u(\Delta)$ (respectively, $\overline{\alpha}_i^l(\Delta)$) has to be transformed to the WAF $\alpha_i^u(\Delta)$ (respectively, $\alpha_i^l(\Delta)$) to indicate the amount of computation time required for the arrived events in intervals. Suppose that the WCET of an event stream is c_i. Then, the transformation can be done by $\alpha_i^u = c_i\overline{\alpha}_i^u$, $\alpha_i^l = c_i\overline{\alpha}_i^l$ and back by $\overline{\alpha}_i^u = \alpha_i^u/c_i$, $\overline{\alpha}_i^l = \alpha_i^l/c_i$.

The WAF provides a more comprehensive bound for guaranteeing the system response time. In some cases, all tasks can meet their deadlines even their occurrences exceed the arrival curve. This is because meeting all task deadlines only requires that the system can provide the needed processing time before the deadline, instead of constraining event occurrences. Therefore, some jobs may leave slacks for other jobs execution if their AETs are less than the WCET. Actually, in most cases, the AETs are less than the WCET.

IV. Execution time feedback scheme

In this section, we first introduce the monitoring algorithm of applying dynamic-counter approach to the system workload monitoring. Then, we analyze the monitoring performance on the system safety.

A. Algorithm

Dynamic counters monitoring is based on the assumption that any monotone and time-invariant event arrival function

Algorithm 1 Implementing the workload-based DC monitor for single staircase function

Input: signal s, $\triangleright tuple < DC_i, CLK_i >$;
1: **if** s = event_arrival **then** $\triangleright$ detect event arrivals
2: **if** $DC_i = N_i^u * C$ **then**
3: reset_timer(CLK_i, δ_i^u)
4: **end if**
5: $DC_i \leftarrow DC_i - C$ $\triangleright$ Be compensated in line 12, 20
6: **end if**
7: **if** s = $CLK_i\_timeout$ **then**
8: $DC_i \leftarrow \min(DC_i + C, N_i^u \cdot C)$
9: reset_timer(CLK_i, δ_i^u)
10: **end if**
11: **if** s = event_finished **then** $\triangleright$ Event finish
12: $DC_i \leftarrow DC_i + C - w_i$
13: **end if**
14: **if** s = idle_entrance **then** $\triangleright$ system become idle
15: reset_monitors
16: **end if**
17: **if** $DC_i < 0$ **then**
18: drop_event
19: report_exception
20: $DC_i \leftarrow DC_i + C$ $\triangleright$ Compensate the loss in line 5
21: **end if**

can be conservatively approximated as the minimum on a set of staircase functions with the form $\bar{\alpha}_i^u(\Delta) = N_i^u + \lfloor \frac{\Delta}{\delta_i^u} \rfloor$:

$$\forall \Delta \in \mathbb{R}_{\geq 0} : \bar{\alpha}^u(\Delta) \leq \min_{i=1..n} (\bar{\alpha}_i^u(\Delta)). \quad (1)$$

As the event streams are scheduled in FP and all event streams are independent, the transformation from arrival curve to workload function is straightforward. If the task has a fixed WCET C for each event, the workload function α^u is $\alpha^u(\Delta) = C \cdot \bar{\alpha}^u(\Delta)$. Therefore, for any monotone and time-invariant event arrival function, its WAF is also the minimum of a set of staircase functions with a stair height C.

The WAF defines a more compact bound for monitoring event trace. In general, the working flow of monitoring with AET feedback is that the WCET is assumed for each arrival event, and the AET is used to replace the WCET after one event is processed. As shown in Fig. 1, the solid curve shows how the runtime workload changes. Every time when new event arrives, the runtime workload will be added the WCET. This workload is corrected by AET feedback whenever an event is processed, as shown in the downward arrow.

The detailed algorithm with the AET feedback scheme is shown in Algo. 1. Instead of event counters in [8], [10], the dynamic counters in Algo. 1 are the allowable burst capacity. Since one dynamic counter (DC_i) can bound only one staircase function ($\bar{\alpha}_i^u$), n dynamic counters are needed to bound n staircase functions. The parameters of monitoring setup are the initial burst N_i^u and the period δ_i^u of the staircase function, and the task WCET C. During the runtime, the system workload is tracked by the dynamic counter DC_i and the timer CLK_i. The timer CLK_i records the time passed within a period δ_i^u. Thus, our monitor maintains 2-tuple $<DC_i, CLK_i>$ during the runtime. The initial value of $<DC_i, CLK_i>$ is $<N_i^u \cdot C, \delta_i^u>$.

The monitoring algorithm is activated by the arrival of the first event. The change of $<DC_i, CLK_i>$ is triggered by signals. The signals can be the arrival of an event, the finish of an event, or the system entrance to the idle state. When an event arrives, all DC_is should decrease C (line 5). After one event is processed, the slack between AET and WCET will be compensated to all DC_is (line 12, where w_i is the AET). When any of DC_is is less than 0, this event is a violation. The system is considered as a lossy real-time system, which means the violated events should be dropped in order to remove the interference on the following events. The detection needs reducing C to all DC_is for every event because the event is assumed to be executed for WCET (line 5 in Algo. 1). But the violated events are dropped, and thus not executed. Therefore, WCET C is added back to all DC_is for every violated event (line 17-21 in Algo. 1), to compensate the loss of the assumed worst-case execution.

The system is assumed to have only two states [10], which are the busy state for event processing, and the idle state for doing nothing. In Algo. 1, whenever the system enters into the idle state, monitors are reset (14-16 in Algo. 1). This is because the task response time does not depend on any task activations before last idle time, and the measured workload before last idle time has no impact on further monitoring.

Theorem 1: Let α^u be a WAF that can make all tasks meet their deadlines if the workload of those tasks does not exceed the WAF. It is sufficient for all deadlines to be met at time $t \geq t_{idle}$ if for all tasks, their workload

$$W(t) - W(t_{idle}) \leq \alpha^u(t - t_{idle}) \quad (2)$$

Theorem 1 is from [13]. From this theorem, we know that the events before last idle time have no impact on the response time of tasks that are activated after last idle time. The system idle state can be checked by checking the pending events. If the system has processed the current event and there is no pending events, the system enters into the idle state.

B. The safety of systems

The proposed monitoring scheme is tight and safe, which means it reduces false negatives and strictly excludes false positives (erroneously accept violated events). To prove the safety of the system, we first state a few definitions.

Definition 1: A busy-time window B is the time interval for the system changing from a busy state to an idle state.

For any event trace, there may be many busy-time windows. Since Algo. 1 is reset whenever system becomes idle, the monitoring in different busy-time windows is independent. If the system is safe in any independent busy-time window, the system is safe in whole time. A general case that can represent any busy-time window is used in this paper. In a busy-time window B_i, the arrival time of an event trace is denoted as $\{t_1, ..., t_n\}$, where t_1 indicates the arrival time of first event, and t_n indicates the arrival time of last event. t_1 is also the start time of B_i. The WCET of every event is the same, and is denoted as C. The AETs of all events in B_i are denoted as $\{e_1, ..., e_n\}$.

Definition 2: The system actual workload $W_a(s,t)$ in an interval $[s,t)$ is the actual needed processing time for processing the events that arrive in the interval $[s,t)$.

The system actual workload is not related to the WCET, but only related to the AET. During the runtime, since the AET of one event is unknown until this event has been processed, the system actual workload cannot be known until the end of one busy-time window. But we can analyze it offline, i.e., the system actual workload can be obtained after all AETs have

been provided. For example, in the busy-time window B_i, if none events are dropped, system actual workload satisfies:

$$W_a(s,t) = \sum_{k=i}^{j} e_k,\ t_{i-1} < s < t_i, t_j < t < t_{j+1}. \quad (3)$$

Similar to the system actual workload, the definition of system estimated workload is as follows.

Definition 3: The system estimated workload $W_{run}(s,t)$ in an interval $[s,t)$ during the runtime is the estimated processing time by the monitor for processing the events that arrive in the interval $[s,t)$.

The workload estimated by our system can be obtained online according to both WCET and AET. As regard to the Algo. 1, the monitor starts to estimate W_{run} at the start of one busy-time window. Therefore, to get the $W_{run}(s,t)$ of Algo. 1, s should be set to be the start time of processing, and t does not exceed over the end of this busy-time window. Take B_i as an example, if none events are dropped by Algo. 1, the system estimated workload satisfies:

$$W_{run}(t_1,t) = \sum_{k=1}^{l} e_k + (j-l) * C,\ t_j < t < t_{j+1}, \quad (4)$$

where l is the number of events that have been processed before the time t.

Note that W_a and W_{run} are different with Eq. 3 and Eq. 4 if there are some events dropped by the monitor. The dropped events are not processed, so W_a and W_{run} should not contain the AET and WCET of the dropped events. Even given a fixed event trace, different monitoring approaches have different W_a and W_{run} because they apply different strategies for dropping events. For example, if Algo. 1 does not introduce the execution time feedback, the estimated AET for every task is WCET, even if the task has been processed. Then, in the case that none events are dropped by this monitor, the estimated workload W^{no}_{run} is:

$$W^{no}_{run}(t_1,t) = j * C,\ t_j < t < t_{j+1}. \quad (5)$$

With the above information, we state the following theorem.

Theorem 2: In any busy-time window, given an event trace R and a WAF α^u, in the case that this trace is monitored by Algo. 1,the system is safe, i.e., the real workload W_a, the runtime estimated workload W_{run}, the WAF α^u satisfy:

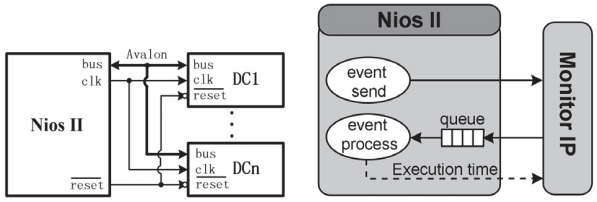
$$W_a \le W_{run} \le \alpha^u. \quad (6)$$

Proof: Consider an arbitrary busy-time window B_i, the arrival trace $\{t_1,...,t_n\}$ in B_i is regulated to $\{t'_1,...,t'_m\}$ by the Algo. 1. At any time t' between t'_1 and t'_m, if there are l' events that have been processed before t', and $t_{j'} \le t' \le t_{j'+1}$, we have

$$\begin{aligned} W_{run}(t'_1,t') - W_a(t'_1,t') &= \sum_{k=1}^{l'} e_k + (j'-l) * C - \sum_{k=1}^{j'} e_k \\ &= \sum_{k=l'+1}^{j'} (C - e_k) \ge 0 \end{aligned}$$

According to Them. 1 in [8], given a trace R and a staircase α^u, in the case that dynamic counter is larger than 0, dynamic counters implementation guarantees that R conform to α^u. The trace R in [8] is an event trace, and the staircase α^u in [8] is arrival curve. This theorem also holds for Algo. 1, i.e., in the case that $DC_i \ge 0$, Algo. 1 conforms the estimated trace workload does not exceed α^u.

Since $W_a \le W_{run} \le \alpha^u$ holds for the whole B_i and the monitoring is independent in every busy-time window, the Algo. 1 can guarantee that the actual workload will not exceed α^u at anytime. Therefore, the system is safety in the monitoring of Algo. 1. ■



(a) Hardware connection (b) Single task monitor system

Fig. 2: Framework of monitors in the system

V. Implementation and evaluation

In this section, we prototype the dynamic counters monitoring as a hardware IP in FPGA, and implement it in scheduling a real-time task in MicroC/OS-II. The implementation platform is Altera DE2 FPGA board. The FPGA is Cyclone IV EP4CE115F29C7N, which is mounted by Nios II processor. The processor frequency is 50 MHz.

A. Implementation

Our monitor is prototyped as an FPGA IP that can be accessed by the Nios II processor. As shown in Fig. 2(a), each DC module monitors one staircase function. Depending on the complexity of the WAF, multiple DC modules can be instantiated for the monitoring. The Avalon bus in Altera FPGA is used to connect DC modules to Nios II processor. In this way, task activations in Nios II can be monitored by calling the function in DC modules. The FPGA IP is compiled in Altera Quartus 12.1. The compilation report shows that, for each DC module, the occupied logic elements are 760, and the occupied registers are 400, which is about 0.5% of the total resource.

A test case is shown in Fig. 2(b). In the Nios II system, message queue model is used in MicroC/OS-II to simulate single task monitoring system. In this system, two tasks are set up. One sends messages to indicate event arrivals, and the other responses to event arrivals for event processing. The monitor is set between the 'event send' task and the queue to detect event arrivals. The accepted events are first stored in the queue, and wait for 'event process' task to fetch. The 'event process' task is activated only when the queue is not empty. When the 'event process' task is activated, the system counts the execution time. For the monitoring with AET feedback, the AET will be sent back to the monitor after an event is processed.

To use the monitor IP, a set of software APIs are defined, which are shown in Tab. I. A simple procedure of using these APIs is shown in Listing. 1. In this example, all DC modules should be configured in the initial phase. Then, whenever task 'event_send' sends an event, DC modules are called to detect the event (Line 1 in Algo. 1). If all returned values are no less than 0, the system accepts the event. Otherwise, this event is

dropped, and the WCET is added back to counters (Line 20 in Algo. 1). After an event is processed, DC modules are called to compensate the loss of assumed WCET by using AET (Line 12 in Algo. 1). If there is no pending event afterwards, all DC modules are reset (Line 14 in Algo. 1).

TABLE I: Dynamic counters API system calls

System call function	Description
monitor_config(base_addr, TOP, period, wcet)	config all modules
monitor_reset(base_addr)	reset all DCi module
event_detect(base_addr)	detect the arrival events
wcet_add(base_addr)	add counter by wcet
aet_write(base_addr, aet)	correct counter by aet

Listing 1: Tasks schedule example

```
void task_initial(void* pdata){    //config the monitor
    monitor_config(base_addr, TOP, period, wcet);
}

void event_send(void* pdata){      //send an event
    flag = event_detect(base_addr);// detect an event
    if (flag == 0){
        drop_event();  // exception and drop violated event
        wcet_add(base_addr); // add wcet to counters
    } else
        OSQPostOpt(msgqueue); //send an event to queue
}

void event_process(void* pdata){ // process an event
    msg = (INT32U *)OSQPend(msgqueue); // obtain an event
    time_start = OSTime; //record the start of a task
    task_execution(); // execute task
    time_end = OSTime; // record the end of a task
    aet_write(base_addr, time_end-time_start);
    // send aet to counters to replace wcet
    if (msgqueue == 0)
        monitor_reset(base_addr); // reset the monitor
}
```

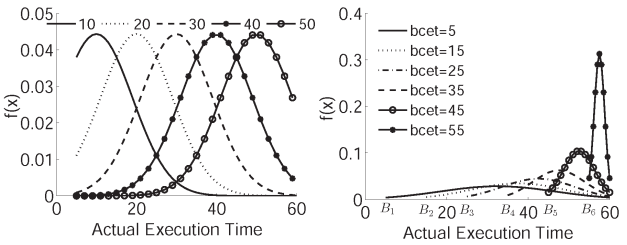
B. Evaluation setup

We compare monitoring results with and without AET feedback. The 'event send' task is set to send an event at a random distance of [20 60]ms. Since 'event send' task acts as an event generator, its timing overhead is negligible and we neglect its execution time on system utilization analysis. The queue size is set to be sufficient large, since we do not consider the case of buffer overflow for this work. If there is no pending event in the queue after one event is processed, the system reset all DC modules. For the case without AET feedback, monitors will not be reset when the system enters the idle state, and *aet_write(base_addr, aet)* will not be called to compensate slacks between AET and WCET.

The arrival curve is assumed as a PJD type with $P = 100$ ms, $J = 300$ ms, and $D = 20$ ms. The WCET is assumed to be 60 ms. According to [11], the arrival curve of the PJD type is the minimum of two staircase functions. Therefore, two DC modules are needed in the monitoring.

In the experiment, two cases are investigated.

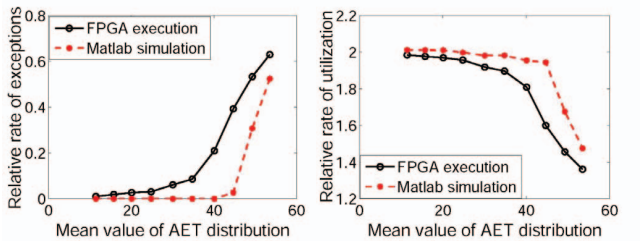
- Case 1: Investigate the impact of AET distribution for a given event trace on monitoring performance by varying the probability distribution of AETs within a fixed [BCET WCET]. In this case, BCET and WCET are set as 5 ms and 60 ms. The AETs are set to conform certain probability distributions. The mean value μ of the distribution can be skewed from 10 ms to 55 ms with a step of 5 ms. Fig. 3(a) only shows μ=10, 20, 30, 40, 50. The standard deviation σ is set as 9 ms for all distributions.
- Case 2: Investigate the impact of the different BCET and WCET pairs on the monitoring performance by varying BCET with a fixed WCET. In this case, the WCET is fixed as 60 ms, and the BCET is set to change from 5 ms to 55 ms with a step of 5 ms. The AETs are set to conform the normal distribution. Fig. 3(b) only shows BCET =5, 15, 25, 35, 45, 55. To be fair, the confidence probability for AETs in [BCET, WCET] is the same as 0.95, i.e., 3.92σ=WCET- BCET. The mean value μ of a normal distribution is (WCET +BCET)/2.



(a) Case 1 (b) Case 2

Fig. 3: Probability distribution of AET with [BCET, WCET]

The AET traces in both cases are generated by MATLAB to conform the specified distributions. Besides, all generated AETs are rounded to be integers.



(a) Relative rate of exceptions (b) Utilization

Fig. 4: The rate of exceptions and utilization of the case 1

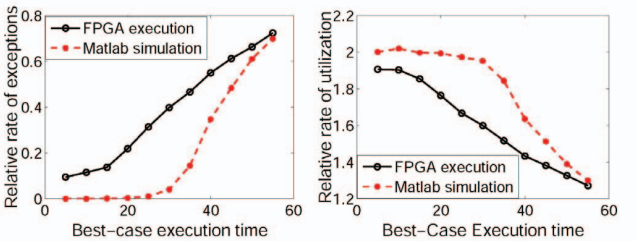
We run each event trace for 60 seconds. During this interval, we count the number of violated events and the system processing time. The relative rate of violated events and processing time between with and without AET feedback are used in the performance comparison. The relative rate of exceptions is that:

$$\frac{\#Violated\ events\ with\ AET\ feedback}{\#Violated\ events\ without\ AET\ feedback}. \quad (7)$$

The relative rate of utilization is that:

$$\frac{System\ processing\ time\ with\ AET\ feedback}{System\ processing\ time\ without\ AET\ feedback}. \quad (8)$$

Furthermore, the optimum monitoring results are obtained form MATLAB simulation. In the MATLAB simulation, the monitor is assumed to be clairvoyant to know the AETs before events have been processed. Then, the actual workload, instead of the estimated workload, is used in the monitoring. Events



(a) Relative rate of exceptions (b) Utilization

Fig. 5: The rate of exceptions and utilization of the case 2

are dropped only if the actual workload exceeds the WAF. In this way, there are no false positives and no false negatives. With the specified event traces and AET distributions as above, the optimum monitoring results within 60 seconds can be simulated by MATLAB.

C. Results

The experimental results are shown in Figs. 4 and 5, where the solid lines and the dashed lines represent the normalized number w.r.t. the no-feedback scheme, for the FPGA execution and MATLAB simulation, respectively. The results reveal that the normalized exception rate is always less than 1 and the normalized system utilization is always larger than 1. This means the AET feedback scheme can reduce the number of the exceptions and improve the system utilization. With more events whose AETs are closer to BCET, i.e., skewed distribution bias on BCET (Fig. 4), the number of exceptions are lower, and the improvement on the system utilization is higher. Similar results can be seen with a smaller difference between BCET and WCET. The reasons are that the slacks are less to be explored for AET feedback scheme in the cases that most AETs are close to WCET (Case 1) or BCET is close to WCET (Case 2).

Another observation is that the performance difference declines faster when the AETs distribution get closer to the WCET. This is because, when the AETs are small, the processor enters into the idle state more frequently, thus resetting the monitors more frequently. Monitors can accept more burst if they are reset. For the monitoring without AET feedback, monitors verify the event trace by employing the arrival curve, so they will not be reset according to the processor state. Therefore, the monitoring with AET feedback performs much better when the BCET is small and most AETs are close to the BCET.

The third observation is that there is no sharp change on the normalized rates for the FPGA execution in the case 2. This is because the distributions of the AETs follow the same distributed within [BCET WCET] . Most of the AETs are concentrated in the mean value of BCET and WCET. Therefore, monitors are not frequently reset even when the BCET is small.

The last observation is that, although the monitoring with AET feedback performs much better than the monitoring without AET feedback, there is still a gap between FPGA execution and the MATLAB simulation. The gap is caused by the difference between the estimated workload and the real workload. Therefore, narrowing the difference between the estimated workload and the real workload is very important in runtime monitoring.

VI. CONCLUSIONS

This paper introduces a workload monitor with the runtime feedback for real-time systems. By using the actual execution time to update the runtime workload, this monitoring scheme can greatly reduce event violations, while guaranteeing all accepted events meet their deadlines. Prototypes running on FPGA show that systems with our monitors can significantly improve the system utilization.

REFERENCES

[1] J.-S. Chenard. *Hardware-based temporal logic checkers for the debugging of digital integrated circuits*. PhD thesis, McGill University, 2011.

[2] A. Daimler Chrysler. Application of software watchdog as a dependability software service for automotive safety relevant systems. 2007.

[3] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the symta/s approach. *Computers and Digital Techniques*, 152(2):148–166, Mar 2005.

[4] 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*, pages 11–20. IEEE Press, 2015.

[5] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. Adaptive workload management in mixed-criticality systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(1):14:1–14:27, 2016.

[6] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. Evaluation and improvements of runtime monitoring methods for real-time event streams. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(3):56, 2016.

[7] B. Hu, K. Huang, G. Chen, and A. Knoll. Evaluation of runtime monitoring methods for real-time event streams. In *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pages 582–587. IEEE, 2015.

[8] K. Huang, G. Chen, C. Buckl, and A. Knoll. Conforming the runtime inputs for hard real-time embedded systems. In *Design Automation Conference*, pages 430–436, 2012.

[9] M. Kim, M. Viswanathan, S. Kannan, I. Lee, and O. Sokolsky. Java-mac: A run-time assurance approach for java programs. *Formal Methods in System Design*, 24(2):129–155, 2004.

[10] K. Lampka, K. Huang, and J.-J. Chen. Dynamic counters and the efficient and effective online power management of embedded real-time systems. CODES+ISSS, pages 267–276, 2011.

[11] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: A hybrid method for analyzing embedded real-time systems. EMSOFT, pages 107–116, 2009.

[12] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.

[13] M. Neukirchner, P. Axer, T. Michaels, and R. Ernst. Monitoring of workload arrival functions for mixed-criticality systems. In *Real-Time Systems Symposium*, pages 88–96, Dec 2013.

[14] M. Neukirchner, T. Michaels, P. Axer, S. Quinton, and R. Ernst. Monitoring arbitrary activation patterns in real-time systems. In *RTSS*, pages 293–302, Dec 2012.

[15] M. Neukirchner, S. Quinton, R. Ernst, and K. Lampka. Multi-mode monitoring for mixed-criticality real-time systems. CODES+ISSS, pages 1–10, Sept 2013.

[16] M. Neukirchner, S. Quinton, T. Michaels, P. Axer, and R. Ernst. Sensitivity analysis for arbitrary activation patterns in real-time systems. In *DATE*, pages 135–140, March 2013.

[17] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems*, volume 4, pages 101–104, 2000.

[18] E. Wandeler. *Modular performance analysis and interface-based design for embedded real-time systems*. PhD thesis, ETH Zurich, Sept 2006.

[19] E. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-Time Systems*, 29(2-3):205–225, 2005.

[20] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *EMSOFT*, pages 80–89, 2005.