Run-Time Monitoring of Timing Constraints: A Survey of Methods and Tools

Nima Asadi, Mehrdad Saadatmand, Mikael Sjödin
Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Västerås, Sweden
nai10001@student.mdh.se, {mehrdad.saadatmand, mikael.sjodin}@mdh.se

***Abstract*—Despite the availability of static analysis methods to achieve a correct-by-construction design for different systems in terms of timing behavior, violations of timing constraints can still occur at run-time due to different reasons. The aim of monitoring of system performance with respect to the timing constraints is to detect the violations of timing specifications, or to predict them based on the current system performance data. Considerable work has been dedicated to suggesting efficient performance monitoring approaches during the past years. This paper presents a survey and classification of those approaches in order to help researchers gain a better view over different methods and developments in monitoring of timing behavior of systems. Classifications of the mentioned approaches are given based on different items that are seen as important in developing a monitoring system, i.e., the use of additional hardware, the data collection approach, etc. Moreover, a description of how these different methods work is presented in this paper along with the advantages and downsides of each of them.**

***Index Terms*—Runtime Monitoring; Extra-Functional Properties; Real-Time; Timing; Survey.**

I. Introduction

The number of computer systems used in our daily life and embedded as part of other systems, such as automobiles, microwave ovens, TV sets, etc., is exponentially growing. The interaction of such embedded systems with their surrounding environments (e.g., through sensors and actuators) often brings along timing requirements. Criticality of these timing requirements, of course, can vary from system to system and under different usage scenarios and situations. Therefore, ensuring that a system respects the timing requirements and operates within the timing constraints defined for it is of great importance and can even determine the success or failure of a computer system (e.g., the airbag system in a car).

While the goal of verification and validation techniques, such as testing, debugging, and theorem proving is to ensure general correctness of programs, the intention of run-time monitoring is to determine whether the current execution meets the specified technical requirements [1]. To achieve this goal, monitors collect the data of interest from the monitored systems, which can be used for further analysis by the user, or the monitor itself.

Timing behavior monitors provide the user with necessary information which can be used to detect or predict violations of timing constraints. Examples of such information are deadline misses and context switches. Many system performance monitoring tools have been developed. However, many of these monitors focus on different aspects of a system performance other than the timing behavior of the system, such as inter-process communications and/or access to shared memory resources. On the other hand, some of the methods used in such monitors are useful in timing performance data collection and analysis as well. Thus, a part of the effort in this paper has been dedicated to distinguishing and including those methods.

Various approaches were suggested in different areas, such as monitoring, debug and replay, and data analysis and visualization. However, the area covered in those studies has mostly been software-fault monitoring in general, i.e., monitors that are used to detect any sort of software fault. The focus in this paper is tried to be on approaches used for monitoring of timing constraints. The data that such monitors provide is especially very important for prediction and analysis of the performance of systems in real-time environment. Our goal is to provide the researchers and developers with a good insight to software monitoring approaches with a focus on timing constraints violation detection. To achieve this goal, an overview of the methods as well as an introduction to the used concepts and definitions is presented. The approaches covered in this study are tried to be a representative sample of timing constraints performance monitoring tools and relative studies.

The organization of this paper is as follows. Section 2 presents a background of the topic as well as the definitions of concepts used in this paper. Section 3 describes the methods used in selecting the monitoring approaches and a brief review of related work. Section 4 discusses the methods, the goal they try to achieve, and the advantages and disadvantages of each method. Section 5 summarizes the survey. The concluding remarks and the future work are given in Section 6.

II. Definitions and Background

Real-time systems are those systems in which the correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced [2]. Although significant work has been done to suggest a method that guarantees the execution of tasks within their pre-specified timing constraints, deadline violation can still happen due to different reasons, such as the unpredictability of the system environment and external signals, and the inability to satisfy all design requirements [3]. Software verification methods are used to make sure that the system meets its general requirements. However, despite the contributions of common verification methods and improvements in real-time scheduling, the need to perform run-time monitoring of these systems is not diminished due to the complexity of these systems and the unpredictability in dealing with the external environment [3]. Therefore, a monitoring tool can be helpful for detecting violations of those timing constraints by collecting, and analysing (depending on the facilities provided by the monitor) relevant system performance data.

According to Peters [4], a monitor is a tool that observes the behavior of a system and determines if it is consistent with a given specification. We decided to use Peters' definition of a monitor, because it covers different categories of monitoring systems. The system in which monitoring, run-time checking, or run-time verification is performed is referred as the 'target system', and the software application whose execution is being monitored is referred to as the 'target application'. The data detected by various monitors can be different. In this paper, our focus is on the data that can be used in analysis of timing behavior of different types of target systems, such as distributed systems, multiprocessor systems, and embedded systems, or information that can help detecting such data. Most of such monitors focus on detecting events of interest. An event is usually a state change in the target application.

1) Latency and interference: Event detection and processing can be performed in different ways, each of them causing a different amount of interference with the target system. The best solution for monitoring with respect to interference is a monitor that uses extra hardware to be able to detect events without affecting the activities of the monitored system. Such tools are usually referred as *hardware monitors*. Run-time monitoring without interference to the target system is usually accomplished by passively monitoring the target processor's data, address, and control buses [5]. *Passive monitoring* is a term used for a type of monitoring that does not affect the target system's performance. However, many state changes of the software being monitored are not reflected by *probes* that are created in data collection lines in the added monitoring hardware. Probes are basically elements of a monitoring system which are attached to the target system in order to collect information about its internal operation. If the internal state of the system context needs to be thoroughly known to make us able to detect an event, a monitoring tool which does not use extra hardware, called *software monitor*, is needed. However, implementing a software monitor needs modification on the target system kernel code, which can alter the behavior of it. To overcome this problem, a *hybrid monitoring* approach could be used. However, this approach that combines the hardware and software monitoring architecture suffers from the same limitations as the hardware monitoring approach. Besides that, the observations will be on a low amount of detail. In order to test and debug a system at satisfactory levels of reliability we need to observe the system completely. We can observe significantly more than it is possible with hardware monitoring approaches by including instrumentation code in the monitored software (application and kernel). Thus, for most application domains, pure software monitoring seems to be the better solution. This can be done by inserting small code stretches in the target program in order to detect events of interest. Different from hardware monitoring systems, software monitoring systems are easier to change. Besides that, the flexibility (modifiability) of software monitoring approach makes it possible to provide more information to programmers and in general, to provide information in a more useful form [5].

As mentioned, including the monitoring code in the target software has the disadvantage of changing its behavior because of the amount of latency being added to it. This is because a part of the CPU time should be dedicated to the monitoring code. This latency is referred as probe effect.

As for the use of the monitoring results, monitors have to choose between a low latency and a small rate of evaluations. Because evaluating a big amount of collected data can increase the latency in the system. The amount of latency usually depends on the focus of monitoring tool and methods. Monitors that only gather data for later use can usually cope with a large latency, whereas monitors that control the monitored system based on the results of evaluations will require a low latency [6].

2) Tracing and Sampling: Data collection can happen in two ways: *tracing* and *sampling*. In tracing, every occurrence of an event creates a record. So event tracing is characterized by the completeness of knowledge [7]. Sampling yields only a statistical measure of the software's execution patterns. It is not precise: if an event does not occur in a sampling log, there is no guarantee that it did not occur in execution. This means that sampling may not be able to detect frequently executed routines whose execution times are smaller than the sampling frequency. However, significantly less time needs to be spent to achieve sampling than to instrument the software system for tracing [7]. Also, the data volume associated with event tracing can be very large. Regarding interference and target behavior change, both event tracing and sampling may affect the performance of the software system. In some literature, tracing is mentioned as *event-driven monitoring* whereas sampling is called *time-driven monitoring*.

III. Survey Methodology

Many run-time monitoring methods have been developed in the past. These methods serve different goals by gathering data of interest from different aspects of systems. Thus, the effort of this paper is to provide an informative categorization of the monitoring tools based on these differences. In this section, the motivation behind this survey and the methods used for the survey are discussed.

A. Objectives of this survey

With ever increasing use of real-time systems, the reliability of such systems seems more and more crucial. In order to make sure that the real-time characteristic of the system is preserved, many techniques for run-time monitoring and debugging of these systems have been developed. In general, monitoring supports the debugging, testing, and performance evaluation of the programs, and covers different aspects of system's behavior, such as memory usage, CPU usage, network and connections status, and tracing of the execution of the processes in the system. Monitoring of timing constraints focuses on the timing behavior of the processes in the system. The main goal in monitoring in this area is to make sure that the real-time quality of the system is guaranteed, which basically means that all the tasks are completed without missing their deadlines. In order to have a solid vision of the performance of a system regarding this quality, it is important that the monitor can provide the data needed for the analysis of timing behavior of the system. The goal of this work is to present an overview of the architecture and workflow of monitors which can be used for timing analysis. A few surveys have been done on run-time monitoring. Moreover, some of the existing works, such as the work of Delgado et al. [1], have tried to cover a wider range of run-time monitors. We tried to narrow down our survey to the monitors whose data can be used in timing analysis of systems. Also, many of the methods covered in this work are not covered in the work of Delgado. The approaches covered in this work are representative samples of such monitoring tools. The categorization that is brought in the summary section is based on the design and architecture of the monitors, the services they provide for the user, and their other important features. Furthermore, the positive and negative points of each method are presented along with the explanation of them to help the readers gain a better vision about them.

B. Related Work

A close work to our work is the taxonomy of run-time software fault monitoring by Delgado et al. [1]. In that work, a classification of tools that monitor software faults is presented. Reinhard Wilhelm et al. [8] discuss the issues in Worst-Case Execution Time (WCET) analysis and review the common suggested tools for this purpose. They divided the tools into two main categories: static methods and measurement based methods. Another survey related to monitoring of system performance is the work of Henrik Thane [9]. Besides an explanation of common concepts and terminologies in performance monitoring, he provides a short review of some of the suggested monitoring methods. In that work, monitors are classified as hardware monitors, software monitors, and hybrid monitors, which are a combination of the first two. A bibliography of the works on performance evaluation was presented by Agajanian in 1975 [10]. Gu et al. provide a review on the literature on monitoring and debugging in their annotated bibliography [11]. They divide their work into four section including modeling and design of the systems, data collection, analysis of the collected data, and dynamic performance controlling. Also, a number of bibliographies of parallel debugging tools were presented by Pancake et al. [12] [13] [14].

C. Review Method

Certain literature review guidelines and approaches were taken into consideration to choose the papers that cover our topic of interest. The application of those approaches is only briefly explained in this section due to space limitations.

a) Inclusion and exclusion criteria: Studies that presented data about software monitoring or performance evaluation were included in the paper data base. The outcome of the studies was not considered in the inclusion criteria. Papers that were published up to 2012 were included in this survey.

b) Search Strategy: The main resources we used to access the papers of interest include the following: ACM Digital Library, IEEE Xplore, ScienceDirect Elsevier, SpringerLink. The main keywords that we used for searching include: Monitoring AND run-time AND software, Performance evaluation AND run -time AND software, Performance Evaluation AND real time AND WCET, Analysis AND run-time AND software, Analysis AND Linux AND run -time, Analysis AND timing constraints. Apart from these main ones, OR combination of some of these keywords were tried and executed as well.

c) Using Citation for Inclusion: Finding the papers that cited a specific paper was the first step of this strategy. Among the papers that were found this way, a number of them were selected according to their relevance to our topic of interest. Specifically, the papers that were about monitoring of irrelevant systems were removed from our survey. For indicating if a paper was relevant or not the whole paper was skimmed or read, because the abstracts would not always give information on whether the paper presented empirical results or not. Another strategy in citation management was to search for the papers cited in the related work section of studied papers. The same relevance check criteria went on for those studies as well.

IV. Overview of the methods

This part presents an overview of the architecture and design of the suggested approaches for the monitoring of system performance regarding to timing constraints. The aspect of monitoring that each work aims to improve or resolve is also stated. Moreover, some of the Pros and Cons of each suggested solution are presented at the end of each section.

A. Non-interference method

1) Objective: to provide a monitoring and debugging system that ensures minimum intervention with the execution of the target system.

2) Approach: The monitor architecture consists of two main parts: the interface module, and the development module [15]. The interface module's major duty is to latch the internal states of the target system based on predefined conditions set by the user. The responsibility of the development module, which contains a general purpose microprocessor, is to start the monitoring process, to record the target node execution history, and to perform analysis on the recorded data. After being connected to a node of the target system and initialized, the interface module keeps collecting events of interest until it finds a stop condition, pre-specified by the user. Then an interrupt is sent to the monitoring processor to separate it from the target processor for the data recording process to take place. The recorded information is transferred to a secondary storage for further processing. The events of interest include process-level events. During the monitoring, the time at which each event happens is recorded. Using this timing information, the execution history can be examined against timing constraint requirements. If violations are found, the replay mechanism can be used to test the program behavior again in order to isolate the errors.

3) Advantages: The monitor imposes low interference to the target system. Also, the start and stop conditions can be planned by the user, which makes this method more flexible.

4) Disadvantages: Generating an interrupt for every event occurrence imposes unpredictable interference to the target system. However, this is the only interference of the monitor with the target system. There is no guarantee that the microprocessor buses of the future will have the properties required to support bus snooping [16], a technique to achieve cash coherence in distributed systems that this type of monitors rely on.

B. PASM

1) Objective: Suggesting a monitor with a flexible specification language to provide the user with automatically defined process-level events to associate them with actions to be taken by a hardware monitoring system.

2) Approach: PASM citeLumpp1 [17] is a programmable hardware monitor, which provides a flexible set of tools for the user to specify events for a wide variety of monitoring applications. The user can include a monitoring section with the application that defines events of interest, actions to be executed upon detection of those events, and the binding of events to actions. This section is then used by the compiler to automatically implement the instrumentation. Events in this monitor are associated with changes of state of the active process. An action can be recording the time of the occurrence of an event to track the timing behavior, or printing of the contents of some internal data structures when a certain point in the execution is reached.

3) Advantages: The programmer has the freedom to define many types of events as functions of the monitored data, and actions corresponding to them. The monitor imposes low interference with the target system. Manual instrumentation is however hard, time-consuming and prone to error, so the automatic instrumentation suggested in this approach removes this problem.

4) Disadvantages: parts of the code, which were affected by the monitoring sections, need to be recompiled when the programmer wants to modify the probes.

C. ART Real-Time Monitor

1) Objective: The objective of ART Real-Time Monitor is to visualize the system's internal behavior with lowest amount of change in its timing behavior.

2) Approach: This monitoring system was developed for ARTS, a distributed operating system developed in 1980 [18] [19] [20]. This approach focuses on visualizing the timing behavior of the system processes. Rate monotonic and deferrable server algorithms are supported by this monitor, and the monitoring task is performed as a part of the target system. The functional structure of the monitoring system can be divided in to three major parts: a part of the target operational system code that records the information of interest about the processes, called event trap, the reporter, which sends the information to the visualizer, and the visualizer, that uses the resources sent from the target system to create historical diagrams of the scheduling decisions of the target system.

An event is generated each time the state of a process is changed. The ARTS monitor records process-level events such as process-creating, waking-up, blocking, scheduling, freezing, killing with completion, killing with missed deadline, and killing with frame overrun [18]. For monitoring timing constraints, the monitor uses the facilities that the ARTS kernel provides, such as 'Time fence'. The 'time fence' is a mechanism in the ARTS Kernel used to detect a timing error at run-time. Before each operation invocation the time fence is checked to verify that the slack time is bigger than the worst case execution time of the invoked operation, and a timer is set. If the execution is not completed within the worst case time, the timer announces an anomaly.

3) Advantages: The integrated scheduler uses rate monotonic scheduling for periodic hard real-time tasks and deferrable server for aperiodic soft real-time tasks. Also, separation between the reporter and the Visualizer makes the monitor suitable for embedded systems.

4) Disadvantages: interactive debugging of real-time systems without deterministic replay is not enough for removing errors because debugging commands can damage the timing-dependent nature of real-time systems [21]. Also, the monitor needs extra kernel support from ARTS, which makes it invasive. If the target system does not provide sufficient resources, the monitoring capability will be limited, consequently, thus, a hybrid approach which uses extra hardware might be necessary.

D. Hmon

1) Objective: To design a transparent monitoring system with continuous data collection facility for HARTS distributed system.

2) Approach: Hmon [21] was developed to monitor the performance of the Hexagonal Architecture for Real-Time Systems (HARTS), a distributed real-time system. The area this monitor covers include monitoring interrupts and shared memory as well as the calls that the users can use in order to monitor the processes that are not covered by the monitor. The monitoring is done by including the monitoring code in the existing system call libraries, meaning that no inside kernel changes are necessary. Context switch events are detected via a hook provided by the pSOS kernel. Task scheduling and CPU usage are determined by studying the order and timing of these events. Also, process management calls, such as process creation and deletion, and time management calls that set or read the clock are recorded to obtain their real-time properties.

3) Advantages: the monitoring is performed transparently, so the programmer does not need to add special code to applications. Also, some system hardware is dedicated to the monitor to minimize interference with the measured system, but no special hardware is required. The system is intended for general-purpose real-time multiprocessors.

4) Disadvantages: data collection code interferes with the system being monitored, and can change the system behavior.

E. Halsall-Hui

1) Objective: to design an interactive monitoring tool for system and application level monitoring that is suitable for embedded systems.

2) Approach: This monitor is designed to gather the data from each processing node of a real-time embedded system which is based on a distributed architecture [22]. The recorded information from each node includes the IDs of the tasks and processes, the type of the tasks and the system calls, and the time that those events happen [22]. The event data recording can take place in two ways. In the first method, the user inserts a library function call, with the corresponding variable name as a parameter, at the appropriate point in the source code, so that whenever that code section is being executed the recording function is called and run. Each of these functions can record a specified event. The event data is then sent to the single monitor of that node, which is also a library function. In the second method, an interrupt is used to periodically refer to a data table provided by the user, which includes event identities, the recording frequency, and the variables to be recorded. This event information is saved in a system file, or an application file, which is downloaded to the monitored system later.

3) Advantages: The method allows application-specific events to be monitored and analysed. The monitor is modifiable, and the monitor interference does not change during replay, which makes the timing behavior of the target system more predictable

4) Disadvantages: This method is invasive and not appropriate real-time systems because of its high amount of interference.

F. Hybrid Monitor

1) Objective: To design a monitor that combines the flexibility of software methods and non-interference of the hardware methods.

2) Approach: This monitor is designed by combining hardware monitors and software monitors [23]. A Test and Measurement Processor (TMP) is integrated to each node of the distributed system in order to record their process and intercommunication activities. The main principle of this method is that the target system generates events of interest, and the TMP hardware processes and time stamps them [23]. The collected event data is stored in a FIFO memory in the CPU. Every time an event is sent into the FIFO buffer the CPU of the TMP is notified by an interrupt activating the processing of the events data. The number of messages, the message length, failed messages, the system time (the time spent for the processes in kernel), and the application time (the time that application processes spend in kernel minus the system time) can be measured using this monitor as well. Events are time stamped locally in this method.

3) Advantages: This method is transparent, i.e., it does not change the behavior of the system, thus the monitoring is continuous. It also uses hardware support to have a low overhead for typical applications. Also, the graphical representation helps better understanding of the recorded data. Furthermore, since the TMPs communicate via their own network, the communication disturbance to the host system is lowered. Another positive point about this tool is that users can load their own evaluation software instead of the default TMP analysis software.

4) Disadvantages: Obtaining hybrid schemes is generally hard. One reason is a lack of architectural support for the monitoring hardware. Standard interfaces are needed to generate industry participation and allow instrumentation portability [24]. Although the analysis part can be changed, till this tool is not flexible for manipulation by the user(for example event detection and type of data being recorded are not decidable). The overhead, although claimed to be small according to the implementation for typical applications (0.1%), is not negligible for real time applications.

G. ZM4

1) Objective: To develop a hardware event driven monitoring tool for parallel and distributed systems.

2) Approach: In this approach, a hardware system called ZM4 , and an event trace processing software called SIMPLE, which works independently from the monitor, are developed [25]. The connection between the hardware and the monitored system is local area network type. Hosting the monitoring system, storing the measured data, and presenting an analysis interface for the users are the responsibilities of the control and evaluation center (CEC) of the ZM4 system. Also, a number of monitor agents are built as slaves for the CEC. Each monitor agent is connected to a target system node. Another responsibility of these probes is time stamping and recording of the events. Time stamping is done using a global clock with the precision of 100 ns. SIMPLE, which works on Linux and MS-DOS, is an evaluation environment used for analyzing the recorded event traces. It generates a global view of the distributed system's behavior and performs trace validation and analysis as well. Whenever the monitor recognizes an event, it stores an event record which consists of event token and a time stamp. The sequence of events is stored as an event Truce [25].

3) Advantages: Distributed hardware monitor ZM4 can be adapted to arbitrary target systems. The combination of event-driven monitoring and event-based modeling makes program instrumentation and validation systematic.

4) Disadvantages: Reliance on event driven monitoring and instrumentation is limited to limit the impact on the target system.

H. Trams

1) Objective: To design a hybrid monitoring method for the performance measurement in both tightly and loosely coupled multiprocessors.

2) Approach: The architecture of this system consists of a software for event triggering, by inserting Write command in the code, and a hardware subsystem used to sample the time and identity of the CPU [26].

For the hardware part, a measurement node consists of a set of VLSI chips with two IC chip types: the Trams (Trace Measurement System) and the Rems (Resource Measurement System). The data written by the user, along with the CPU identification and the time stamp are stored in the Trams sample memory. The sample memory then reads this information for further analysis. Rems is used for data sampling. The target distributed system can be tightly coupled or loosely coupled. In the first system a single node can be used in a centralized event trace collection, and in the second architecture model each node can be connected to a corresponding processor. Both the Trams and Rems contain three sections: a data capture system, an output, and a FIFO buffer.

3) Advantages: Both loosely coupled and tightly coupled systems are covered in this approach. As a special feature, event counters are implemented in one of the VLSI chips in order to reduce the amount of data to be transferred and evaluated [26].

4) Disadvantages: The monitoring tool is system specific, and it makes it easy to generate so much data that it swamps any file system or data analysis station [27].

I. Alamo

1) Objective: A method to reduce development costs for a broad class of execution monitors.

2) Approach: Lightweight Architecture for Monitoring (Alamo) [28] [29] [30] is an event-driven monitor developed for C programs, and uses the Icon programming language to specify assertions. The Alamo monitoring architecture utilizes CCI, a Configurable C Instrumentation tool as a preprocessor that uses parse trees to identify monitoring points and inserts events into the target program source code. The architecture of this monitor consists of: (1) an automatic instrumentation mechanism, (2) an execution model, (3) abstractions for event, selection, multiplexing and composition, and (4) an access library that allows monitors to directly manipulate target program state. Alamo employs automatic program instrumentation to produce target program events for the monitor. The Execution Monitor (EM) executes the Target Program (TP) and then returns control with information in the form of an event report. The user can apply a predicate to each event report to make monitoring more specific, or view detailed information through Alamo's visualization mechanism.

3) Advantages: The Alamo monitor architecture significantly reduces the development cost of writing program execution monitors

4) Disadvantages: There is no support for real-time or shared-memory multiprocessor-based parallel applications. Not all execution monitors can be written using an Alamo-based framework; those that, cannot tolerate intrusion of instrumentation code require a two-process model such as that employed by standard source-level debuggers [31].

J. MAC

1) Objective: To propose a tool that complements testing (infeasible to completely test the entire system due to the large number of possible behaviors), and verification (possibilities for introduction of errors into an implementation of a design that has been verified) techniques.

2) Approach: Monitoring and Checking (MAC) [32], [33], [34], [35], [36], [37] provides a framework for runtime monitoring of real-time systems written in Java. The MAC architecture consists of three main components: a filter, and event recognizer, and a run-time checker. The filter, which maintains a table containing names of monitored variables and addresses, extracts low-level information, time stamps it, puts it in a message, and sends it to the event recognizer. From this low level data, the event recognizer detects the occurrence of abstract requirement level events based on the Requirement specifications written in Meta Event Definition Language (MEDL), and informs the run-time checker about them. The run-time checker uses these events to see if the current system execution conforms with the requirements of the system. An event is an instantaneous state change. Static analysis is used to determine monitoring points, which are inserted automatically.

3) Advantages: The filter (that extracts the information of interest) is separated from the event recognizer, so that system execution does not suffer from the overhead of abstracting out events from low-level information. This architecture is also appropriate for monitoring distributed systems where each module is able to have a corresponding filter.

4) Disadvantages: This architecture adds to the communication overhead because the filter sends the data to the event recognizer. the executing software needs to send enough state information to observer process, in order to check constraints and do analysis. When violation of the constraints happens, observer process cannot stop the execution of the software.(there is no feedback to the system), but this is feature is added in MACS, a later work [33].

K. PMMS

1) Objective: To minimize the total time between formulation of the questions (what the monitor should do) and delivery of the answers. The second is to minimize the monitoring overhead during execution.

2) Approach: Program Monitoring and Measuring System (PMMS) [38]. is a monitoring approach that automatically collects high level information about the execution characteristics of a program. Data collection is done by code inserted into the source program of the target system, and conditions are used to filter out events that are not relevant. The monitor handles events of interest by installing code that reacts whenever they occur. This data collection code includes Pre-condition (relevance test), Local-variables (used to store local data), Before-code (code to collect data available before the event), After-code (code to collect data after the event), Post-condition(a relevance test based on data that is available after the event), and Action(code that stores data more permanently for later use) [38]. Examples of recorded event data include the time at which the event occurs, the value of program variables at that time, etc. The user can specify all the objects and relations using the high level specification language that is provided in this method. The PMMS uses a main memory for the active database to facilitate the collection, computation, and access to the computation results.

3) Advantages: The used specification language in this work allow security engineers to write a centralized policy specification; the systems then uses a tool to automatically insert code into untrusted target applications. This centralized policy architecture makes reasoning about policies a simpler and more modular task than the alternative approach of scattering security checks throughout application or execution-environment code. With a centralized policy, it is easy to locate the policy-relevant code and analyze or update it in isolation [39].

4) Disadvantages: Since the instrumentation code performs database queries, instrumentation can significantly change the performance of the target program.

L. JRTM

1) Objective: An approach for monitoring timing constraint violations in real-time systems. The objective is to detect timing violations as early as possible.

2) Approach: Java Runtime Timing-constraint Monitor [40], [41] targets timing properties of distributed, real-time systems written in Java. In this work, the necessary constraints and event log are automatically derived by the compiler, and then the compiled specification is loaded into the monitor at run-time. Java programmers can insert the event triggering method calls in their Java programs where event instances are supposed to occur. At run-time, whenever an event method is executed, the current system time is recorded as the event occurrence time and this timestamp is sent to the monitor along with the event name. The monitor keeps these event occurrence messages in a sorted queue with the earliest event message at the head of the queue. The event message at the head is processed at an appropriate time to check it with the related constraints. Once a violation of the specification is found, users are notified. This monitor can run on the same machine as the target process or on a standalone monitoring machine.

3) Advantages: Low overhead; it uses small size of event record history depending on the maximum occurrence rate of events.

4) Disadvantages: It is difficult to timestamp an event with an accurate time point, which is assumed to be measured well for JRTM to use.

M. GRTMon

1) Objective: To design a run-time monitor with small probe effect, and no input missing (not for non-real-time purposes).

2) Approach: Generalized Run-Time Monitor (GRTMon) [42] is a tool for real-time systems to detect information regarding timing constraints. In this method, data collected by sensors is written to buffers from which monitors read. Each buffer is mapped at the respective sensor section and all associated monitor tasks. According to the work flow of this monitor, data pairs of an output element and its timestamp are the input to evaluation algorithms of the monitor. Monitors sort the buffer output elements based on their timestamps before evaluation. The CPU's timestamp counter, which contains the number of elapsed CPU cycles since the CPU has been initialized is used by the sensor to tag the output with its corresponding timestamp. A sensor directory is used to provide relations between sensors and monitors. Thus, there is no direct relation between sensors and the monitor, which can be effective in decreasing the probe effect of the monitor. The monitor can either run as a constant-bandwidth server with a bandwidth that the user defines, or resource requirements can be determined based on the sensors' jitter-constrained stream specifications [42]. Also, in GRTMon, monitors and the target system communicate asynchronously, so the monitors have less direct influence on the monitored system's timing. Examples of events of interest are context switches, inter-process communication (IPC) or events in the kernel itself such as calls to certain kernel functions.

3) Advantages: Using this method evaluation of events with least amount of input data miss is guaranteed. Also, small set of dependencies between the monitor and the target system and sensors and the monitor decreases the overhead on the target system.

4) Disadvantages: If more than one sensor is used the overhead will increase significantly.

N. FKT

1) Objective: To design a simple software monitor for Linux with lower interference which can support multiprocessor platform and networked environment.

2) Approach: Fast Kernel Tracing (FKT) [43] monitor is a software tool designed to evaluate the performance of Linux kernels running on Pentium PCs. This monitor is implemented by modifying the Linux kernel through adding probes for data collection, and user-level programs for data evaluation. The probes are placed by the programmers. By default probes are placed at the entry to and exit from every system call, trap, interrupt, and process switch inside the kernel [43]. The timing recorded by a probe is the time provided by the Intel Pentium's timestamp counter which is incremented on every hardware clock cycle. The data recorded by the probe consists of the time at which the data is recorded by the probe, a unique identification code assigned to the probe, the ID of the current process, the number of the processor, and additional parameters provided by the programmer. The monitor has two phases: recording, which happens during the run-time, and analysis, which happens off-line. The analysis part can be changed by the user for different types of evaluations. Also, the information to be collected can be specified by the programmer while inserting the probes.

3) Advantages: The probes can be turned on and off using a key mask that is controlled by user-level programs, so that the probing overhead is reduced when probes are not needed to be used. Also, the amount of information recorded by each probe is small, which means that big traces of operating system execution can be recorded.

4) Disadvantages: This tool does not provide a run-time analysis of data, so the user does not notice violation occurrence during the run-time. When the buffer is filled the probing is suspended, which implies the use of a big buffer.

O. SoC-based Monitor

1) Objective: A runtime monitor within an embedded system to detect timing specification violations

2) Approach: The System on Chip-based monitor [44] uses a hybrid method for run-time verification of embedded systems. The monitor consists of event recognizer, a verification tool, and the monitor output. The event recognizer decides if the collected data is relevant to the event definition. After passing this step, the event data is sent to verification section where it is compared with the requirement constraints. In case a violation is observed, it is sent to the output of the monitor. The events detection code is inserted in the source code of the target system, but no code is needed for transmitting the events to the event recognizer. In fact, the event data is transmitted from the target system to the event recognizer by a dedicated monitoring core called 'event dispatcher' [44].

3) Advantages: Low overhead due to use of extra hardware for event dispatching. It benefits from a light design for monitoring of embedded systems.

4) Disadvantages: Limited monitoring is available due to the memory constraints of embedded systems. The monitor's performance is highly dependent to the target system hardware specifications.

P. Raju-Jahanian

1) Objective: Early detection of violations of timing assertions in an environment in which the real-time tasks run on multiple processors

2) Approach: This monitoring tool consists of a set of cooperating monitor processes one on each processor of the target system [3]. Upon occurrence of an event, application tasks on a processor inform the local monitor by putting the event into a queue in shared memory. Then, a monitor process decides whether the event must be communicated to other monitors or not. The role of this monitor is to make sure the violation is predicted as early as possible [45], by deciding if the data is communicable or not using intermediate constraints. The main idea behind this solution is that 'it is possible that an implicit constraint is violated before an explicit delay or deadline becomes unsatisfiable at run-time' [45]. If the occurrence time of an event has to be sent to a remote monitor, the monitor puts the event and its local occurrence time into a message and sends it to other monitor processes. If a message arrives from a remote monitor or a timeout occurs, a monitor checks if violation has occurred. If a violation is detected, it notifies the application task (with termination as the default action).

3) Advantages: The intermediate monitor makes early violation detection possible

4) Disadvantages: It uses Real-Time Logic specification language (RTL) for constraints and event-action bonding, which is rarely used in practice.

Q. OSE Monitor

1) Objective: To facilitate the possibility of monitoring of timing behavior for OSE real time operating system.

2) Approach: The main idea behind this approach is to add a second layer scheduler to the OSE (Operating System Embedded) real-time operating system to make it easier to query the execution result of real-time tasks [46]. This adjunct scheduler uses the specifications of real-time tasks, such as the period and execution time of each task, from a parameter file. According to these parameters the second layer scheduler schedules the tasks by allowing them to be sent to the core scheduler in Earliest Deadline First(EDF) or Rate Monotonic Scheduling (RMS) scheduling algorithms. Thus, it is clear that the second layer scheduler process must have the highest priority among all the OSE processes.

The monitor process works with the lowest priority, i.e., as a background OSE process, in order to make sure that it does not interfere with the scheduling process. Upon completion of a task, the monitor receives a signal from the second layer scheduler. Two types of log files are created in this process: a scheduling log file, and a monitoring log file. The scheduling log file, which is created by the second layer scheduler, contains the time points at which a task in the task set is scheduled, completed, or preempted. Monitoring log file, which is created by the monitor, is updated only when an instance of a task is completed [46].

3) Advantages: A very good set of timing information is provided by the log files without further analysis processes, which makes this tool very easy to use.

4) Disadvantages: Dynamic creation of tasks is not covered in this method. The overhead of another scheduling layer on the real-time system can be significant.

V. SUMMARY

A number of suggested tools were selected out of a bigger group of studies on system monitoring and performance evaluation. As mentioned before, the focus of this paper is on the tools and methods whose presented data can be used for timing analysis of the system performance. Other

TABLE I
A CLASSIFICATION OF MONITORS

Approach	Monitor Adaptibility		Data Collection		Design Method			Development stage		Target System		
	Specific	General	Tracing	Sampling	Hardware	Software	Hybrid	Research	Production	Real-Time	Embedded	Distributed
Non-inter	x			x	x			x		x		x
PASM	x		x		x			x				x
ART			x		x			x		x		x
HMON	x			x		x		x		x		x
Halsall-Hui		x	x	x		x		x		x	x	x
Hybrid		x	x				x	x				x
ZM4		x	x				x	x				x
Trams		x	x				x	x				x
Alamo	x		x			x			x			
MAC	x		x			x			x	x		
Pmms		x	x			x		x				
JRTM	x		x			x		x				x
GRTMon		x	x			x		x		x		
FKT	x		x			x		x				
Soc-based		x	x				x	x		x	x	
Raju-Jahanian		x	x			x		x		x		x
OSE monitor		x	x			x		x		x		

performance evaluation approaches such as debugging, testing, and visualizing were not covered in this survey.

In this section, a classification of the reviewed tools is provided in Table I. This classification is based on the features that can be useful in giving the developers and researchers a broad insight on different suggested approaches in designing system performance monitors. These features were chosen in order to satisfy the goal of facilitating the process of research on run-time monitoring of timing properties for the readers. A description of each classification element is provided in the sections below

A. Monitor Adaptability

Depending on the design purpose, some of the monitoring tools are developed for a specific target system. In many cases, the architecture of such monitors is dependent to the facilities that the target system provides. Monitors that are not designed for a specific target system can provide the developers the possibility of designing transparent monitoring for target systems with basic facilities and source code in any programming language. In our classification, 'General' adaptability means that the monitoring method can be used for different types of target systems. We chose the term 'specific' for the tools that were developed for a specific target system, or monitor programs in a specific programming language, and is not not possible to be implemented for other systems.

B. Data Collection Method

An important task of any a run-time monitor is to collect the data of interest from the monitored system when it is running. Two types of data collection during the system execution are sampling and tracing. A brief description of the two mentioned methods was previously given.

C. Design Method

As explained in the prior sections, depending on the use of extra hardware in the monitoring system, a monitor can be hardware, software, or a combination of the two, called hybrid. A description of the advantages and drawbacks of each type is given in the previous sections.

D. Development Stage

While some of the covered methods were employed in software production projects, thus are available tools, the others are classified as research project prototype.

E. Target System

As mentioned in previous sections, the monitors covered in this work are designed for different environments and platforms of target systems ranging from embedded systems to distributed and parallel systems. This section on the table represents the type of target systems that the monitors were designed for, or can be used for. Some monitors, such as FKT, were designed for general-purpose systems.

VI. CONCLUSION

There is an increasing need in monitoring of timing behavior in different types of computer systems. This is mainly because of the growing importance of the issue of satisfying timing constraints in many systems that are being used today, particularly embedded devices. A practical and reasonable method for controlling a system's timing behavior is through run-time monitoring of timing in the system. In this paper, we provided a survey of a selected group of works on monitoring of timing constraints in different systems and contexts. The systems in need of monitoring covered in this work ranged from embedded systems to hard real-time and distributed systems. Our main intention with this work has been more to gather versatile monitoring contexts and methods than merely analyzing monitoring methods targeted for a single specific context or monitoring methods using the same design architecture (both in terms of hardware or software implementation). For each approach that was covered, a review of its work flow and design of each was presented as well as their advantages, drawbacks, and the problem each of them aim for. Then, a short summary and a classification of the methods were offered based on the each method's architecture and other practical features.

Software and hardware monitors have been developed to tackle different monitoring needs and to enable collection of data considering the interference of the monitor in the target system's performance, which is referred to as probe effect. In this sense, hardware monitors try to minimize the interference and performance penalty of monitoring, while software monitors generally provide a more flexible and customizable solution. Also, hybrid monitors have been designed as a combination of the two mentioned architectures in order to resolve their issues, and benefit from the advantages of each. However, due to the complicated nature of timing behavior of systems, and the increasing complexity of different systems, adaptation and customization of existing methods may be required to match the needs of different systems and contexts. Hence, this paper's effort in summary has been on giving system designers and developers an organized insight toward the important available experiences in this area. This is achieved by not only describing different monitoring methods for different contexts, but also providing a classification framework for them.

VII. Acknowledgement

This work has been partially supported by the Swedish Knowledge Foundation (KKS) through the ITS-EASY industrial research school [47], and also by Combitech [48] and Xdin [49] companies in Sweden.

References

[1] N. Delgado, A. Q. Gates, and S. Roach, "A taxonomy and catalog of runtime software-fault monitoring tools," *IEEE Trans. Software Eng.*, pp. 859–872, December 2004.

[2] J. A. Stankovic and R. K., "What is predictability for realtime systems," *Springer*, November 1990.

[3] S. C. V. Raju and F. Jahanian, "Monitoring timing constraints in distributed real-time systems," in *Proc. Real-Time Systems Symp.*, vol. 30, pp. 57–67, 1992.

[4] D. K. Peters and D. L. Parnas, "Requirements-based monitors for real-time systems," *ISSTA '00, ACM Press.*, December 2002.

[5] S. Ricardo and J. R. De Almeida, "Run-time monitoring for dependable systems: an approach and a case study," in *in Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS 2004)*, vol. 30, pp. 41–49, October 2004.

[6] T. Riegel, "A generalized approach to runtime monitoring for real-time systems," *Master's thesis, TU Dresden*, 2005.

[7] E. Metz, R. Lencevicius, and T. F. Gonzalez, "Performance data collection using a hybrid approach," in *Proceedings of the 10th European software engineering conference held jointly with 13th ACMSIGSOFT international symposium on Foundations of software engineering*, vol. 30, pp. 126–135, 2005.

[8] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstrom, "The worst case executiontime problem, overview of methods and survey of tools," *Trans. on Embedded Computing Sys.*, 2008.

[9] H. Thane, "Testing and debugging of distributed realtime systems," *PhD Thesis, Mechatronics Laboratory, Royal Institute of Technology, Stockholm, Sweden*, May.

[10] A. H. Agajanian, "A bibliography on system performacne evaluation," *Computer*, November 2000.

[11] W. Gu, J. Vetter, and K. Schwan, "An annotated bibliography of interactive program steering," *ACM SIGPLAN Notices*, September 1994.

[12] S. Utter, C. M. Pancake, and K. Schwan, "A bibliographyof parallel debuggers," *ACMSIGP1un Notices*, pp. 29–42, November 1989.

[13] C. M. Pancake and S. Utter, "A bibliographyof parallel debuggers," *ACMSIGP1un Notices*, pp. 21–37, January 1991.

[14] C. M. Pancake and R. H. B. Netzer, "A bibliographyof parallel debuggers," in *Proc. of the 3rd ACM/ONR Workshop on Parallel and Distributed Debugging, San Diego, CA, USA*, May 1993.

[15] J. J. P. Tsai, K. Y. Fang, and H. Y. Chen, "A noninvasive architecture to monitor real-time distributed systems," *Computer*, pp. 11–23, March 1990.

[16] M. M. Gorlick, "The flight recorder: An architectural aid for system monitoring," in *Proc. ACM/ONR Workshop Parallel and Distributed Debugging*, pp. 175–183, 1991.

[17] H. J. Siegel, T. Schwederski, J. T. Kuehn, and N. J. Davis, "An overview of the pasm parallel processing system," *in Tutorial, Computer Architecture*, pp. 387–407, 1987.

[18] H. Tokuda, M. Kotera, and C. W. Mercer, "A real-time monitor for a distributed real-time operating system," in *Proceedings of ACM SIGOPS and SIGPLAN workshop on parallel and distributed debugging*, May 1988.

[19] H. Tokuda and M. Kotera, "A real-time tool set for the arts kernel," in *Proceedings of 9th IEEE Real-Time Systems Symposium*, December 1988.

[20] H. Tokuda, M. Kotera, and C. W. Mercer, "An integrated time-driven scheduler for the arts kernel," in *Proceedings of 8th IEEE Phoenix Conference on Computers and Communications*, March 1989.

[21] P. S. Dodd and C. V. Ravishankar, "Monitoring and debugging distributed real-time programs," *Software Practice and Experience*, pp. 863–877, October 1992.

[22] F. Hasall and S. C. Hui, "Performance monitoring and evaluation of large embedded systems," *Software Engineering Journal*, pp. 184–192, 1987.

[23] D. Haban and D. Wybranietz, "A hybrid monitor for behavior and performance analysis of distributed systems," *Software Engineering Journal, IEEE Trans. Software Eng.*, pp. 197–211, February 1990.

[24] C. Alexander, "Multicomputer performance monitoring: a standards-based approach," *Technical Report MSSU-EIRS-ERC-93-13 Mississippi State University*, December 1993.

[25] R. Hofmann, R. Kar, B. Mohr, A. Quick, and S. M., "Distributed performance monitoring: Methods, tools, and applications," *IEEE Trans.Parallel and Distributed Systems*, 1994.

[26] A. Mink, R. Carpenter, G. Nacht, and J. Roberts, "Multiprocessor performance-measurement instrumentation," *Computer*, pp. 63–75, September 1990.

[27] J. K. Hollingsworth, B. P. Miller, and J. Cargille, "Dynamic program instrumentation for scalable performance," in *Proc. Scalable High-Performance Computing Conference, Knoxville, Tenn.*, pp. 841–850, 1994.

[28] C. L. Jeffery, "Program monitoring and visualization: An exploratory approach," *Springer-Verlag*, 1999.

[29] C. L. Jeffery *New Mexico State Univ., Las Cruces, N.M., personal comm.*, 2002.

[30] C. L. Jeffery, W. Zhou, K. Templer, and M. Brazell, "A lightweight architecture for program execution monitoring," in *Proc. ACM SIGPLAN/SIGSOFT Workshop Program Analysis for Software Tools and Eng.*, pp. 67–74, 1998.

[31] C. L. Jeffery, "The alamo execution monitor architecture," *Electronic Notes in Theoretical Computer Science*, 2000.

[32] M. Kim, S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan, "Java-mac: A run-time assurance tool for java programs," in *Proc. Fourth IEEE Intl. High Assurance Systems Eng. Symp.*, pp. 115–132, 1999.

[33] M. Kim, I. Lee, and O. Sokolsky *Univ. of Pennsylvania, Philadelphia, personal comm.*, 2002.

[34] M. Kim and M. Viswanathan, "Mac: A framework for run-time correctness assurance of real-time systems," *Technical Report MS-CIS-98-37, Dept. of Computer and Information Sciences, Univ. of Pennsylvania*, December 1998.

[35] M. Kim and M. Viswanathan, "Formally specified monitoring of temporal properties," in *Proc. European Conf. Real-Time Systems*, 1999.

[36] I. Lee and H. Ben-Abdallah, "A monitoring and checking framework for run-time correctness assurance," in *Proc. 1998 Korea-U.S. Technical Conf. Strategic Technologies*, 1998.

[37] I. Lee and M. Kim, "Runtime assurance based on formal specifications," in *Proc. 1999 Int. Conf. Parallel and Distributed Processing Techniques and Applications*, 1999.

[38] Y. Liao and D. Cohen, "A specificational approach to high level program monitoring and measuring," *IEEE Trans. Software Eng.*, pp. 969–978, November 1992.

[39] J. Ligatti, "Policy enforcement via program monitoring," *Ph.D. thesis, Princeton University*, 2006.

[40] A. Mok and G. Liu, “Efficient run-time monitoring of timing constraints,” in *Proc. Third IEEE Real-Time Technology and Applications Symp.*, pp. 252–262, 1997.
[41] M. Moller, “Runtime assurance based on formal specifications,” *Univ. of Oldenburg, Oldenburg, Germany, personal comm.*, 2002.
[42] T. Riegel, “A generalized approach to runtime monitoring for real-time systems,” *Master’s thesis, TU Dresden*, 2005.
[43] R. D. Russell and M. Chaven, “Fast kernel tracing: A performance evaluation tool for linux,” in *Proceedings of the 19th IASTED International Conference on Applied Informatics (AI 2001), Innsbruck, Austria*, February 2011.
[44] C. Watterson and D. Heffernan, “A monitoring approach to facilitate run-time verification of software in deeply embedded systems,” *Doctoral thesis, University of Limerick, Ireland*, March 2010.
[45] S. C. V. Raju, R. Rajkumar, and F. Jahanian, “Monitoring timing constraints in distributed real-time systems,” in *Proc. Real-Time Systems Symp.*, pp. 57–67, 1992.
[46] M. Saadatmand, M. Sjodin, and N. Ul Mustafa, “Monitoring capabilities of schedulers in model-driven development of real-time systems,” *17th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, Sepember 2012.
[47] ITS-EASY post graduate industrial research school for embedded software and systems. http://www.mrtc.mdh.se/projects/itseasy/, Accessed: September 2013.
[48] Combitech. http://www.combitech.se//, Accessed: September 2013.
[49] XDIN AB. http://xdin.com/en/about-xdin/enea-experts/, Accessed: September 2013.