Software Active Online Monitoring Under Anticipatory Semantics

Changzhi Zhao, Wei Dong, Ji Wang, Ping Sui and Zhichang Qi
National Laboratory for Parallel and Distributed Processing, P.R.China
Email: john9908009@gmail.com ,{dong.wei, ji.wang}@263.net

***Abstract*—As the increment of software complexity, traditional software analysis, verification and testing techniques can not fully guarantee the correctness and faultlessness of deployed systems. As a result, software health management has been proposed as an effective complement of traditional quality assurance methods. Runtime monitoring is critical for software health management. Runtime verification is one of the monitoring techniques based on formal method, but it only generates the passive monitor and cannot predict violations until they occur. This paper presents the ongoing work of software active online monitoring via looking ahead into a runtime partial system model to explore the possible fault states in advance. The procedures include monitoring the current execution, predicting the occurrences of violations based on anticipatory semantics, and preventing them by intervening system's execution. The formal description of active online monitoring problem and a corresponding architecture are further discussed in the paper.**

I. Introduction

The correctness and dependability of software are very important in safety-critical systems. In such systems, unintentional design, implementation and runtime faults might result in injury or even death to human being. To make sure that the software systems are really safe, a lot of methods have been extensively studied and applied such as verification and testing. However, these efforts still cannot assure that the system will function correctly in runtime. Formal verification dedicates to check all possible executions of the system, but the analysis usually works well on software's abstract models, and will meet the state explosion problem for complex systems or software program. Software testing may obtain good effects for software implementation, but does not guarantee that all behaviors are analyzed. Another important problem for these methods is the runtime environment can not be completely predicted during software development, such as for the systems in space missions. When the software complexity increases, it becomes exceedingly hard to exhaustively test and verify the software, with the result that latent faults might still remain in the software.

From above views, the health management of deployed software is necessary in runtime. The concept of software health management (*SHM*) came from the counterpart "Integrated Systems Health Management "which had been widely adopted in complex heterogeneous physical systems. The idea is that the health of systems should be continuously monitored, and if anomalies are detected, their source will be isolated and appropriate mitigation actions will be taken.

Monitoring of the running system is critical for *SHM*. Software monitoring can be classified into online monitoring and offline monitoring [1]. In the setting of *SHM*, online monitoring is necessary in order to tune the behavior of system in time if needed. Runtime verification is one of the important methods for software monitoring, and also suits online monitoring. It has been designed to verify whether the trace of the software system is consistent with its requirements when the implementation is executing in actual context. It is expected that online runtime verification could pave the way for not only detecting incorrect behavior of a software system, but also for reacting and potentially healing the system. But current runtime verification techniques only generate passive online monitors, which can not predict violations in advance and such that can not prohibit the occurrence of failures. One important reason is that they are not based on the anticipatory semantics and do not concern with the information about the system model [2]. We think that predicting the faults before actually occurring is more valuable for *SHM*. It gives system a chance to pre-tune its behavior and prohibit the occurrence of the failures. Therefore, the runtime verification should not be passive but active.

In this paper, inspired by the above idea, we present our ongoing work about software active online monitoring which dedicate to improving the traditional runtime verification to active monitoring. Its purpose is not repairing the fault after it has been detected, but predicting the faults in advance and triggering the control actions to prevent the software from failures. We found that limited knowledge of the system model might be sufficient to predict violations with high confidence. The process of active online monitoring will exploit such limited system knowledge to predict violations. Active online monitoring is not only analyzing the current event but also looking ahead into a partial system model to explore the state space in advance. If non-conformance has been detected, correspondence control actions will be taken to prevent the system from reaching a violation.

II. Anticipatory semantics of monitor

In active online monitoring, the triggering of control actions will be based on the verdict of runtime verification, thus the monitoring semantic is vital important. In fact, because of the importance of temporal logic in critical systems, we aim at deriving a verdict whether there might be any infinite executions dissatisfied a correctness property by considering the finite prefix. To capture implication of this idea, a monitor should follow two maxims: *Impartiality* and *Anticipation*.

Traditional runtime verification is based on the finite trace semantics, it does not obey these two maxims. So [3] proposed the three-valued semantic for runtime verification. Given the monitored property φ, based on the infinite trace semantic of temporal logic formula and the finite trace which has been observed so far, the result of monitoring should be: *true*, $false$, or *inconclusive*. For every prefix $\pi \in \Sigma^*$, if two infinite continuations $\sigma, \sigma^{'}$ exist such that $\pi\sigma \models \varphi$ and $\pi\sigma^{'} \not\models \varphi$ hold, the semantic $[\pi \models \varphi]$ evaluates to the inconclusive verdict ?. On the other hand, once only satisfied or only unsatisfied continuations exist, the semantic $[\pi \models \varphi]$ evaluates to the corresponding verdict *true* or $false$. The definition is as follow, and an uniform approach for synthesizing monitors for linear logic is provided in [4].

$$[\pi \models \varphi] = \begin{cases} true & if \quad \forall\sigma \in \Sigma^\omega : \pi\sigma \models \varphi; \\ false & if \quad \forall\sigma \in \Sigma^\omega : \pi\sigma \not\models \varphi; \\ ? & otherwise. \end{cases}$$

Intuitively, the three-valued semantics is suitable for software active online monitoring. Only if the verdict is $false$, corresponding control actions need to be triggered; only if the verdict is *true*, the monitor can terminate; otherwise, the monitoring process should continue.

III. Software Active Online Monitoring

A. The Problem

Software active online monitoring is more than runtime verification. It can be defined as the process of analyzing partial system model that looks ahead to detect nonconformance (prediction) and applies control actions to the system (prevention). To handle the worst cases, we assume that the active monitor and the monitored system will reside in different machines and executing concurrently. n is the round-trip communication delay between the active monitor and the monitored system [5]. We also assume that the processing delay of the active monitor is negligible compared with the communication delay.

Firstly, we formulate the model of the monitored system M_s as a transition system $M = < S, s_0, T, \Sigma >$, where S is the state space, s_0 is the initial state, Σ is the event set which are controllable by the monitor, and T is the transition function $T : S \times \Sigma \rightarrow S$. In more detail, the construction and working process is as follow. At the beginning, a set of property-relevant operations (e.g., a set of system calls relevant to the desired property) is identified. Then, all the sequences of these operations that are allowed in normal will be found, and be encoded with transition system M. Meanwhile, any execution of a program defines a trace, which is a sequence of property-relevant operations performed during that execution. The transition system M is used to monitor program execution: as the program executes a property-relevant operation, M reaches to a new state. It is required that the M is an exact model, i.e. $L(M_s)$=$L(M)$, where $L(M_s)$ is the set of all feasible sequences for property-relevant operations, and $L(M)$ is the language accepted by M.

Then we formulate the m-step partial model of the system which originates from the current state as the runtime model [2][6][7]. If the communication delay between the monitor and the system is n, then $n + 1$ is the minimum amount of looking ahead required to ensure that control actions will be received by the system in time. Formally, the runtime model for current state s'_0 and the system model M can be defined as $M_{s'_0} = < S', R, \Sigma, s'_0, F, C, T' >$ which is a tree-like structure with bounded depth. S' is the set of the runtime states, R is a function which maps runtime states in S' to static states of M. $s'_0 \in S'$ is the initial state of $M_{s'_0}$ and Σ is the event set. $T' : S' \times \Sigma \rightarrow S'$ is a transition function defined on $M_{s'_0}$ such that $R(T'(s', e)) = T(R(s'), e)$ where $e \in \Sigma$. $F \subseteq S'$ is the set of final states of the runtime model within $n + 1$ steps, and $C \subseteq S'$ is the set of the control states defined as $C = \{s' \in S' \mid \exists e \in \Sigma, f \in F, T'(s', e) = f\}$. The control actions for s'_0 will disable some of the transitions with the source states in C.

Software active online monitoring can then be decomposed into the problem that given the communication delay n, the system model M, and the current state s'_0:

(1) Generate the runtime model $M_{s'_0}$ which originates from state s'_0. At the initial, s'_0=s_0.

(2) Assumed that π is the finite trace which has been observed so far, $\sigma \in \Sigma^*$ is any path originating from s_0 in the runtime model, φ is the user defined property. It will check whether $F' = \{\sigma \in \Sigma^* \mid [\pi\sigma \models \varphi] = false\}$ is empty. If not, F' is returned.

(3) Generate appropriate control actions to ensure that the system does not reach violated states by restricting the runtime model, i.e. disabling the transition set $E = \{e \in \Sigma \mid s' \in S', f \in F', T'(s', e) = f\}$.

(4) Determine the mechanism for executing the control actions in the system.

B. The Architecture

In terms with the problem described above, we present the architecture of active online monitor in Fig.1, which is under anticipatory semantics. In the architecture, (1) represents the events which is generated from instrumented system during runtime; (2) represents a set of final states $F = \{f_1, ..., f_n\}$ of runtime (partial) model, where f_i is identified by a finite state trace from the initial state of partial model; (3) is another set of final state $F' = \{f'_1, ..., f'_m\}$, which will violate the property specification based on anticipatory semantics of runtime verification; and (4) is the corresponding control actions obtained from the partial model to prevent the system from reaching a violation. At the end, in terms with the control actions which generated from the partial model, the controller send signals to the system such that the behavior of the system would be steered in time when necessary.

The architecture of active online monitor is composed of the controller, the anticipating runtime verifier, the partial model generator and the control action generator. They interact with

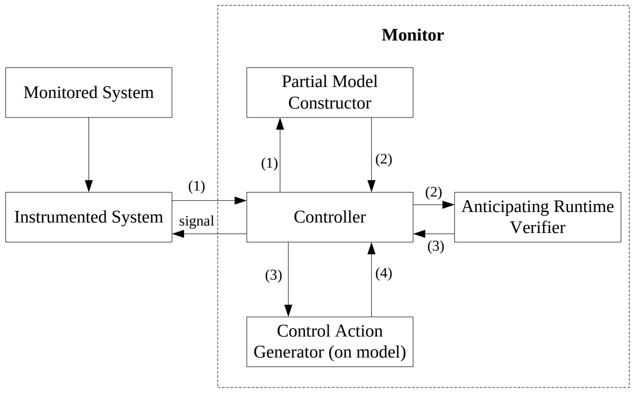
Monitor
Monitored System
Partial Model Constructor
Instrumented System
Controller
Anticipating Runtime Verifier
Control Action Generator (on model)
signal
(1)
(2)
(3)
(4)

Fig. 1. The architecture of active online monitor

the instrumented system. The functions of these components are described below.

Instrumented System: The system is instrumented in terms of the monitored property. When one of the monitored states is changed, an event is generated, the instrumented system sends an event notification to the controller. Once a steering action is received, the system executes the accompanying control actions.

Controller: On receiving of an event, it transfers the event to partial model generator; on receiving of a set of final states $F' = \{f'_1, ..., f'_m\}$, it transfers the set to anticipating runtime verifier; on receiving of a set of violation states, it transfers the set to control action generator; and on receiving of the corresponding control actions, it send signals to the system such that the running of the system will change appropriately in time.

Partial Model Generator: After receiving an event notification from the controller, partial model generator generates the partial runtime model M_{s_0} based on the previous runtime model. If $e \in \Sigma$ is the current event received from the system, the partial model generator update its model state to the new system state $s'_0 \in T'(s_0, e)$. It then computes the state $f \in F \backslash F'$ of $M_{s'_0}$ which are reachable from s'_0. So the partial model is generated by determining the successor state of the reachable set using system model M. Once the partial model has been generated, the new final state set F will be returned to the controller.

Anticipating Runtime Verifier: The monitor is generated using the method of runtime verification under anticipatory semantics like in [4]. The monitor will check whether any state in F violate the user-defined property. After each checking, the current state must be recovered for next checking. Finally, it reports the set F' of all violation states and the set of all satisfied states to the controller.

Control Action Generator: On receiving the set of violations F', for each state $u \in C$ which has transitions to F', control action generator will generate the set Σ_d^u of events that need to be disable. The control action for the state s_0 is then given by $\Sigma_d = \cup_{u \in C} \Sigma_d^u$. On receiving the set of satisfied states, it will send the state set to the system. If the running of the system reaches one of these states, the system will send a special event notification to the controller such that the monitoring process can be terminated.

IV. Conclusion and Future Work

With the knowledge about design model of the system and current finite execution trace, software active online monitoring based on anticipatory semantics will be able to predict occurrence of violations well in advance and enable the system away from these violations via steering. In this paper, we assumed that these events are controllable, which is not always true. Thus, we should further study how to make the system steerable. In our continuing work, we will also study how to make the active online monitoring more efficient and widely applicable.

Acknowledgment

The work is supported by National Natural Science Foundation of China under Grant No.60673118, No.60725206 and No.60803042.

References

[1] Martin Leucker and Christian Schallhart. A brief account of runtime verification. Journal of Logic and Algebraic Programming, Elsevier B.V., 2008

[2] Arvind Easwaran, Sampath Kannan, and Oleg Sokolsky. steering of discrete event systems: control theory approach. In Electronic Notes in Theoretical Computer Science. volume 144,Issue 4(21-39), 2005.

[3] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In Foundations of Software Technology and Theoretical Computer Science. LNCS 4337, Springer-Verlag, 2006.

[4] Wei Dong, Martin Leucker, and Christian Schallhart. Impartial anticipation in runtime verification. In Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis(ATVA'08), LNCS 5311, Springer-Verlag,2008.

[5] Christos G.Cassandras, Stephane Lafortune. Introduction to Discrete Event Systems. Kluwer Academic Publishers, 1999.

[6] Sheng-Luen Chung, Stephane Lafortune, and Feng Lin. Limited lookahead policies in supervisory control of discrete event systems. In IEEE Transactions on Automatic Control, volume 37, IEEE, 1992.

[7] Sheng-Luen Chung, Stephane Lafortune, and Feng Lin. Supervisory control using variable lookahead policies. In Proceedings of Discrete Event Dynamic Systems. Kluwer, 1994.