Reducing trace size in multimedia applications endurance tests

Serge Vladimir Emteu Tchagou, Alexandre Termier, Jean-François Méhaut, Brice Videau, Miguel Santana, René Quiniou

Reducing trace size in multimedia applications endurance tests

Serge Vladimir Emteu Tchagou
University of Grenoble Alpes
Email: serge-vladimir.emteu-tchagou@imag.fr

Alexandre Termier
University of Rennes 1
Email: alexandre.termier@irisa.fr

Jean-Franois Méhaut
University of Grenoble Alpes
Email: jean-francois.mehaut@imag.fr

Brice Videau
University of Grenoble Alpes
Email: brice.videau@imag.fr

Miguel Santana
STMicroelectronics
Email: miguel.santana@st.com

Ren Quiniou
Inria Rennes
Email: rene.quiniou@inria.fr

***Abstract*—Proper testing of applications over embedded systems such as set-top boxes requires *endurance tests*, i.e. running applications for extended periods of times, typically several days. In order to understand bugs or poor performances, execution traces have to be analyzed, however current trace analysis methods are not designed to handle several days of execution traces due to the huge quantity of data generated. Our proposal, designed for regular applications such as multimedia decoding/encoding, is to monitor execution by analyzing trace on the fly in order to record trace only in time periods where a suspicious activity is detected. Our experiments show a significant reduction in the trace size compared to recording the whole trace.**

I. Introduction

Nowadays, there is a huge market for electronics consumer products such as smart-phones, tablets or set-top boxes. These products are powered by MultiProcessor System-on-Chip (MPSoC), which are highly integrated chips having on a single die several generalist computation cores, specialized accelerators, memories and I/O components. Demanding applications such as 4K video decoding require to exploit all the processing power of MPSoC through a parallel approach. However parallel programming is notoriously difficult and even small timing errors between components may lead to low QoS of the applications. In such cases, traditional debuggers cannot be used as they are extremely intrusive: they will change the timings of components execution and hide potential errors. The solution used in industry is to continuously collect *traces* of the application execution through dedicated, low-intrusive tracing hardware. These traces are analyzed post-mortem in order to understand bugs or reasons for poor performance.

Industrial validation of the applications requires "endurance tests" (typically several hours/days of continuous execution) in order to verify that bugs do not occur after long period of time. Tracing such tests would generate hundreds of GigaBytes of traces, which are difficult to analyze with state of the art approaches based on visualization [1] or data-mining [2]. Hence currently these tests do not benefit from tracing, whereas they exhibit some of the most complex bugs to detect.

Based on the observation that most applications tested are multimedia applications, which exhibit regular behaviors, our proposal (Section II) consists in capturing only parts of the trace from long running tests that contain problematic execution periods (if any) and that can be analyzed by existing tools. For this purpose, we use machine learning to model correct behavior during the monitoring of the application and record the trace only when the detected behavior departs significantly from the correct behavior. Using such approach on real execution traces, our experiments (Section III) show that trace size can be significantly reduced.

II. Methodology

In this section, we describe our main contribution: an online approach to analyze the trace stream of a multimedia application, and record only suspicious portions of the trace. This approach is based on *anomaly detection* techniques from the data mining field.

Such techniques require two steps: first a learning step in order to model the correct behavior from a reference execution trace. Then a monitoring step where incoming trace data is compared to that model. If the difference is too large, the corresponding trace data is recorded to a storage device.

Data representation: In order to perform these steps, an adequate trace representation has to be found. A raw trace is a sequence of timestamped events. In a streaming context, trace data is not provided event by event by the tracing hardware, but by *windows* of N consecutive events. The value of N is usually correlated to the size of the buffers of the tracing hardware. Such window will be our elementary processing unit.

Each window is transformed as a *probability mass function* (pmf), i.e. a vector giving for each event type the number of occurrences of that event type in the window. Such vectors are easy to manipulate and give a good abstraction of the trace in a window.

Learning: Our approach exploits the Local Outlier Factor (LOF) [3] anomaly detection method, which is designed to handle such *pmfs*. The learning step of LOF is simple and requires only a *reference trace* for a correct execution of the application. This trace is divided in windows, and the pmf of each window is computed. This gives a set of points in an high dimensional space, which constitutes the model of the reference trace. Depending on objectives of the endurance tests planned, a reference trace can be simply the trace of the first few minutes of application execution, during which

the developer noticed no QoS errors. A curated database of reference traces can be constituted in order to skip the learning step.

Online anomaly detection: Once the model is constructed, the anomaly detection step can take place. As trace windows arrive, their pmf representation is computed. The algorithm evaluates the similarity between N_{pmf}, the pmf vector of the current window, and P_{pmf}, the pmf vector associated with past windows. We use Kullback-Leibler distance [4] because it is well suited to compare pmf vectors. If N_{pmf} and P_{pmf} are similar, we consider that no significant change occurred, no anomaly detection tests are performed. In this case N_{pmf} is merged with P_{pmf} and P_{pmf} is updated: this helps to detect slow changes of behavior.

On the contrary, if P_{pmf} and N_{pmf} are dissimilar, then a LOF computation is performed on N_{pmf} in order to determine if the new window is an anomaly. The idea of the LOF computation is to place N_{pmf} in the space of high dimensional points found during the learning set, and to compare the density of points around N_{pmf} with the density of points around the K nearest neighbors of N_{pmf}. If N_{pmf} has the same density than its neighbors (LOF = 1), then it is embedded in a cluster of "regular" points and does not represent an anomaly. If N_{pmf} has a lower density than its neighbors (LOF $\geq \alpha > 1$), then it is not in a cluster of "regular" points and is likely to represent an anomaly. α is a user given parameter of the approach.

If an anomaly is detected, the corresponding window of trace is recorded to a storage device.

III. Experiments

In this section, we demonstrate the applicability of our approach on a real application by monitoring its trace in a controlled setting. In this preliminary experiment, the application monitored is GStreamer, a well known video decoding framework. GStreamer is run on a standard laptop with an Intel core i7 processor X 920 @ 2.00GHz and is limited to using one core. It is tasked to decode a video of 6h17m. Its trace output is monitored by our prototype. Other parameters are: windows size of 40ms, $K = 20$.

The first 300s of decoding are used as a reference in order to learn a trace model for the correct behavior. Then every 3 minutes, a "perturbation" is made during 20s through an heavy processing application. The goal of the experiment is to verify that our monitoring approach records trace mostly for windows impacted by the perturbation. Note that due to the Gstreamer buffering mechanism, the perturbation impact on the video is delayed (Δ_s) from start of the perturbation and the end of the perturbation impact is delayed (Δ_e) from the return to the normal video playback. The status of playback can be deduced from error messages sent by GStreamer. We computed a mean value Δ_s^{avg} and Δ_e^{avg} on a two minutes part of the video in order to fix these experimental parameters. During monitoring, each trace window W monitored is labeled as: *True Positive (TP)* if it is in $[Perturbation_{start} + \Delta_s^{avg}, Perturbation_{end} + \Delta_e^{avg}]$ and GStreamer reports an error and $LOF(W) \geq \alpha$, *False Negative (FN)* if it is in $[Perturbation_{start} + \Delta_s^{avg}, Perturbation_{end} + \Delta_e^{avg}]$ and GStreamer reports an error and $LOF(W) < \alpha$, *False Positive (FP)* if $LOF(W) \geq \alpha$ and either GStreamer does not report an error or the window is not in $[Perturbation_{start} + \Delta_s^{avg}, Perturbation_{end} + \Delta_e^{avg}]$, *True Negative (TN)* in the other cases.

The quality of anomaly detection is evaluated with the well known metrics of $precision = TP/(TP+FP)$ and $recall = TP/(TP+FN)$. $precision$ gives the ratio of windows that are correctly reported as anomalous, the higher its value the higher the reduction rate of the trace. $recall$ gives the ratio of actually anomalous windows found by our approach, a low value means that some potentially buggy parts of the trace were missed by our approach.

The values of $precision$ and $recall$ according to the LOF threshold α are given in Figure 1.

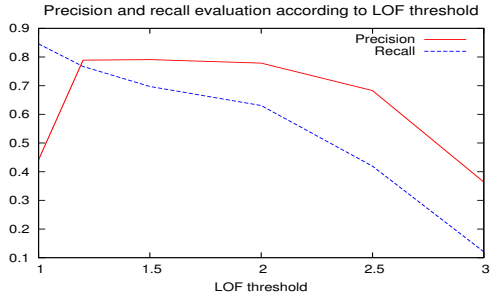

Fig. 1. Precision and recall of anomaly detection

With $\alpha = 1.2$, the precision is 78.9% and the recall is 76.6%: these values indicate that most anomalies were correctly detected. In this setting, the recorded trace weights $418MB$, instead of $5.9GB$ if recorded completely: this represents a 14-fold reduction in volume. This confirms the interest of our approach.

IV. Conclusion

We have presented an approach for online monitoring of multimedia applications in endurance tests, in order to reduce the volume of trace recorded and ease further debugging. Experiments show a reduction of one order of magnitude in the recorded trace size, with a good coverage of the anomalies happening in execution. We plan to apply our approach on larger scale endurance tests of actual embedded systems. We are also interested in further reducing the recorded trace size by exploiting the periodic behavior of the application.

References

[1] B. Cornelissen, A. Zaidman, D. Holten, L. Moonen, A. van Deursen, and J. J. van Wijk, "Execution trace analysis through massive sequence and circular bundle views," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2252–2268, 2008.

[2] S. Lagraa, A. Termier, and F. Pétrot, "Scalability bottlenecks discovery in mpsoc platforms using data mining on simulation traces," in *DATE 2014, Dresden, Germany, March 24-28, 2014*, 2014, pp. 1–6.

[3] M. M. Breunig, H. peter Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings 2000 ACM SIGMOD International Conference On Management of Data*, vol. 29, no. 2, 2000, pp. 1–12.

[4] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, pp. 79–86, 1951.