Distance-based Trace Diagnosis for Multimedia Applications: Help me TED!

Christiane Kamdem Kengne*†, Noha Ibrahim*
*University of Grenoble,LIG
681 rue de la passerelle
38400 Saint Martin d'Hères
France
{surname.name}@imag.fr

Marie-Christine Rousset*, Maurice Tchuente†
†University of Yaounde I,LIRIMA
BP 812 Yaoundé, Cameroun
UMI 209 UMMISCO
BP 337 Yaoundé, Cameroun
m.tchuente@uy1.uninet.cm

***Abstract*—Execution traces have become essential resources that many developers analyze to debug their applications. Ideally, a developer wants to quickly detect whether there are anomalies on his application or not. However, in practice, the size of multimedia applications trace can reach gigabytes, which makes their exploitation very complex. Usually, developers use visualization tools before stating a hypothesis. In this paper, we argue that this solution is not satisfactory and propose to automatically provide a diagnosis by comparing execution traces. We use distance-based models and conduct a user case to show how TED, our automatic trace diagnosis tool, provides semantic added-value information to the developer. Performance evaluation over real world data shows that our approach is scalable.**

***Index Terms*—Execution traces; Distance; Diagnosis; Audio/Video decoding; Multimedia applications.**

I. Introduction

With the proliferation of embedded systems providing an everywhere access to multimedia contents, the development of multimedia applications is an area of high competition in which, the time lost by a developer to debug the application amounts a financial loss for companies.

The analysis of execution traces, that are sequences of time-stamped events, is at the core of the optimization and debugging of applications. When the developer has a reference trace (which can be produced by a simulator), a technique for detecting possible anomalies within an execution trace is to compare it with the reference trace using a suitable distance [1]. However, although there is an abundant literature about distances on sequences ([2]–[4]), very few distances take into account the temporal aspect that is crucial in execution traces. In fact, designing an appropriate distance for a meaningful comparison between multimedia execution traces is a difficult task. Indeed, it requires to capture and combine within a single numerical function, several aspects that are specific to multimedia execution traces. Whatever the quality of a distance for suggesting the existence of a bug in an execution trace, based on its numerical comparison with a reference trace, the results of the distance calculation are inherently difficult to interpret by human developers, in particular for finding the actual cause of the bug.

In this paper, we propose to replace a black-box approach encapsulated in a single complex distance by a glass-box approach based on a fine-grained analysis of problems that are likely to occur in multimedia applications. The idea is that anomalies in multimedia applications usually have visible effects (for users) such as desynchronization between sound, picture or subtitles, the interruption of a video streaming or the loss of some frames (a frame being an image rendered during a known time interval).

We make the following contributions:

1) We have identified a family of anomalies likely to occur in multimedia applications and that are visually perceptible when a user is watching a video.
2) For each type of anomaly, we have designed a specific distance which measures appropriately the amplitude of the corresponding anomaly.
3) Based on these distances, we have designed a diagnosis tool able to detect degraded execution traces and to identify the causes of such a degraded behaviour.

The rest of the paper is organized as follows: Section II provides some background and states the problem of automatic diagnosis. In Section III we present the general approach that we propose to solve this problem. In Section IV, we describe our TED tool and illustrate it on a use case. Section V summarizes experiments conducted using TED. Finally, Section VI concludes with some perspectives.

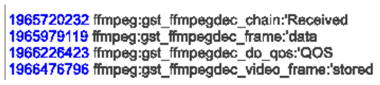
II. Background and Problem Statement

In this section, we explain how execution traces are obtained and described as timestamped sequences. We also introduce three well-known types of anomalies occurring in video streaming. Finally, we state the trace diagnosis problem.

A. Execution Traces Generation and Description

Embedded systems directly integrate hardware tracing support to collect events generated by applications or perform a post-mortem analysis of their execution. These techniques minimize intrusiveness, i.e tracing an application has a minimal impact on its behaviour, allowing complex interactions to be shown in real-time applications such as video decoding.

Based on our previous work [5], we formalize the execution traces that are generated as sequences of timestamped events as depicted in Fig. 1, where, for instance, 1965720232 (in ms) is a timestamp of the event ffmpeg:gst_ffmpegdec_chain:'Received.



```
1965720232 ffmpeg:gst_ffmpegdec_chain:'Received
1965979119 ffmpeg:gst_ffmpegdec_frame:'data
1966226423 ffmpeg:gst_ffmpegdec_do_qos:'QOS
1966476796 ffmpeg:gst_ffmpegdec_video_frame:'stored
```

Fig. 1. An execution trace

B. Audio/Video decoding Anomalies Description

While streaming a video, some common anomalies can occur. These anomalies are well known in the community of A/V developers ([6,7]) and almost always have visual and sound effects on the video streaming. They can even be simulated using existing tools that are able to inject those perturbations. We have chosen to detect three of the common anomalies:

P_1**: Audio/video/subtitle desynchronization anomaly**: This anomaly reflects a desynchronization in time between audio, video or subtitles. The audio may be slower than the video or the subtitle may not appear at the right moment.

P_2**: Player crash anomaly**: The player stops abruptly at a random execution time, without any reason.

P_3**: Slow streaming anomaly**: Visually, video is very slow. In this case the audio/video/subtitles are synchronized but take much more time than in a normal execution.

C. Trace Diagnosis Problem Statement

The general trace diagnosis problem that we consider is to detect whether an execution trace presents some anomalies reflecting an abnormal behaviour of the application under supervision, and if this is the case, identify the cause or at least the type(s) of these anomalies.

This problem is difficult to solve in general, i.e. without exploiting some additional knowledge or without restricting its scope in order to exploit some domain-specific characteristics. Our approach is to exploit error-free *reference* traces that can be obtained by a simulator, and to compare them with real execution traces using suitable distances. A simulator is a tool usually used by developers, in order to restore the good video decoding environment. Detecting whether a real trace execution is abnormal consists in a distance-based comparison with the reference trace obtained by the simulator ran on the same video and identifying pre-established types of domain-specific anomalies, namely those mentioned in Section II-B and referred to as P_1, P_2 and P_3 respectively.

The trace diagnosis problem that we consider in this paper can then be stated as follows:

Given an execution trace T and a reference trace T_r, how to automatically detect whether T contains anomalies of type P_1, P_2 or P_3, using a distance-based comparison with T_r.

III. Distance-based diagnosis

In this section, we explain our general approach for solving the trace diagnosis problem stated above, using appropriate distances.

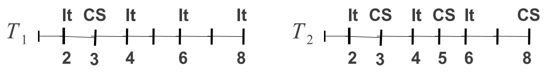
A distance d between two objects is a numerical measure of how far apart these objects are [8]. Instead of defining a single distance as a black-box to detect various anomalies, our glass-box approach defines multiple distances that are appropriate to the types of anomalies we want to detect.

The procedure that we followed is: First, we decode a movie video with $gstreamer$ and obtain a reference trace. Then, we inject in the streaming, perturbations corresponding to the three types of anomalies and we obtain for each anomaly the corresponding abnormal execution traces. Finally, for each type of anomaly, we manually analyze the reference trace and the execution trace, and extract the differences that are relevant for each distance. We defined three distances that we present briefly. The detailed formalization and algorithm computation of these distances can be found in [9].

A. Occurrence distance

For P_1 anomaly, when examining the traces, one can detect different numbers of occurrences of some events in the simulated trace and the abnormal one. We define the *occurrence distance* between two traces as the number of events whose ratio of occurrence in the two traces is less than a given threshold. This distance is appropriate to retrieve P_1, A/V/S desync. anomaly, because it measures the number of events that differentiate T_1 from T_2.

Example 1: consider the traces T_1 and T_2 below, and the threshold $\theta = 0.5$. Event *It* occurs 4 times in T_1 and 3 times in T_2. Its occurrence ratio is $3/4 = 0.75 > 0.5$. Event *CS* has a ratio of $1/3 = 0.33 < 0.5$. Thus it is the only event with occurrence ratio less than θ, then $d_1(T_1, T_2) = 1$.

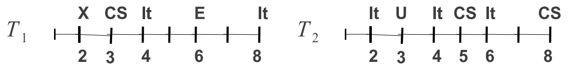


B. Dropping distance

For P_2 anomaly, when comparing the simulated and abnormal traces, we found that some events seem to appear only in one trace and not in the other one. The corresponding *dropping distance* refers to the number of distinct events that belong only to one trace.

This distance is appropriate to retrieve P_2, i.e. Player crash anomaly (see section II-B).

Example 2: for the traces T_1 and T_2 below , $events(T_1) = \{$X, CS, It, E$\}$; for T_2; $events(T_2) = \{$CS, It, U$\}$; $events(T_1) - events(T_2) = \{X,E\}$ and $events(T_2) - events(T_1) = \{U\}$, then $d_2(T_1, T_2) = |\{$X, E, U$\}| = 3$.

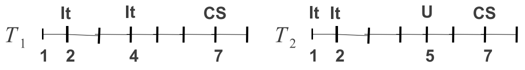


C. Temporal distance

For P_3 anomaly, the duration and the order of some events differ in the two traces. In the abnormal trace, some events durations are much longer than in the simulated trace.

The temporal distance that we propose is an adaptation of the distance model of *Mannila et. al* [10] which is an edit-distance taking into account temporal aspects. It uses 3 distinct operations (Insertion, Deletion, Move) with a cost associated to each of it, in order to evaluate the total cost of transforming one sequence to another. The distance between two traces T_1 and T_2 is the cost of the cheapest sequence that transforms one sequence into the other.

Example 3: For traces T_1 and T_2 below, the cheapest order-preserving sequence of operations that transforms T_1 into T_2 is $Move(It, 2, 1), Move(It, 4, 2), Ins(U, 5)$.



However, the beginning timestamp in two traces is not always the same. Consequently, results obtained with this method are not satisfactory. Therefore, we adapt the Mannila distance model in order to have $d_3(T_1, T_2) = 0$ when T_2 is obtained from T_1 by a time shift (for details, see [9]).

IV. The TED tool illustrated on a use case

In this section, we describe TED, our TracE Diagnosis tool (Fig. 2), and illustrate its functioning on two use cases.

A. TED Architecture

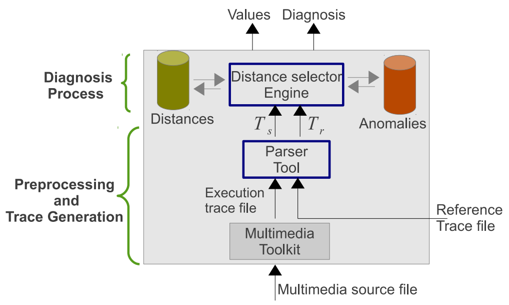


Fig. 2. TED Architecture

TED handles two main phases. In the *Preprocessing and trace generation* phase, the Parser tool takes as input an execution trace generated from a multimedia source file via the *multimedia Toolkit*, and a reference trace file. The traces are preprocessed. This step is very important for a successful outcome of the analysis as a non cleansed and non normalized data can lead to spurious and meaningless results [2]. A parsed trace T_s (respectively T_r) is obtained from execution trace (respectively reference trace), by removing some redundant information or by modifying others.

The *Diagnosis process* is the second and core phase of TED. The *distance selector engine* selects an appropriate distance from the *Distances* database and applies it to the T_s and T_r.

B. Use cases

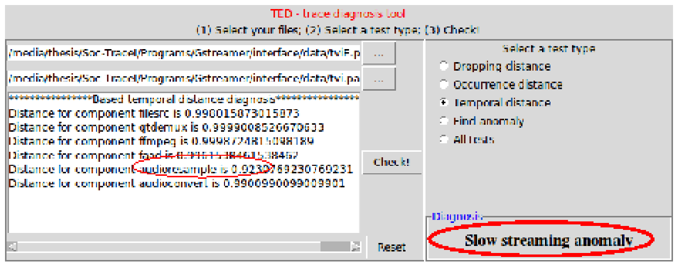
We consider the following scenario. A developer retrieves several execution traces of video streaming watched by a user and needs to know if there exists anomalies, and what are these anomalies.

In the *Preprocessing and Trace Generation* phase, we decode the movie with *gstreamer* to obtain the reference trace T_r. We use a gstreamer element *identity* [11], with property *sleep-time*, to obtain a A/V/S desync. anomaly (scenario b). The abnormal trace obtained is T. We generate another abnormal trace, with a slow streaming anomaly (scenario a) by a stress of CPU and memory in the system.

In the *Diagnosis process* phase, the developer uses TED as follows:

Case 1: he has an idea of the anomaly and just want to verify if his hypothesis is true or not. He selects the distance to apply and TED gives the diagnosis. In Fig. 3, *temporal distance* is used (scenario a). The developer suspects a slow streaming anomaly (P3). TED detects the anomaly and returns the value of temporal distances between the two traces per plugins. TED points out the a plugin to be the one with the most dissimilar events between the two traces.

Case 2: he has no idea of what is happening and would like to find if there exists an anomaly in T. He selects the choice *all distances*, and TED applies successively all the distances.


TED - trace diagnosis tool
(1) Select your files; (2) Select a test type; (3) Check!

/media/thesis/Soc-Trace/Programs/Gstreamer/interface/data/tviF.p
/media/thesis/Soc-Trace/Programs/Gstreamer/interface/data/tvi.pa

```
***************Based temporal distance diagnosis***************
Distance for component filesrc is 0.990015073015073
Distance for component qtdemux is 0.999900852667063
Distance for component ffmpeg is 0.999872481 5088189
Distance for component faad is [illegible]
Distance for component audioresample is 0.923076923076923
Distance for component audioconvert is 0.9900990099009901
```

Select a test type
( ) Dropping distance
( ) Occurrence distance
(•) Temporal distance
( ) Find anomaly
( ) All tests

Diagnosis
Slow streaming anomaly

Fig. 3. A *slow streaming anomaly* detected (scenario a)

By using TED, a developer analyzing an execution trace is notified of anomalies with their types and locations in the trace (the plugin concerned). TED is a time saver for developers as they can quickly detect anomalies in execution traces.

V. Experiments

We conducted a set of experiments to demonstrate the quality and efficiency of our proposed execution trace diagnosis tool. We show how helpful this automatic tool can be for developers, by an evaluation of TED scalability and precision.

System configuration: Our prototype system is implemented in Python 3.2. The experiments were run on an Intel Xeon E5-2650 at 2.0GHz with 32 Gigabytes of RAM with Linux.

Data Set: We use traces from two real applications, described below:

Gstreamer application: Gstreamer [11] is a powerful open source multimedia framework for creating streaming applications, used by several corporations such as Intel, Nokia and STMicroelectronics. For these experiments we decoded several movies using Gstreamer on a Linux platform, with the $ffmpeg$ plugin for video decoding.

GSTapps application: It is a test video decoding application for STMicroelectronics development boards. The execution trace contains both application events and system-level events. It is generated from a $ST40$ core of the SoC, which is dedicated to application execution and device control.

Table I gives a description of reference traces.

Running time and Scalability: Fig. 4 reports the wall clocks of TED for *occurrence* and *dropping* distance, when varying the number of events in execution traces. The horizontal axis represents the maximum number of events of the two

TABLE I
EXPERIMENTAL DATASET

Video	Duration	Nb. of events	Size
generic	$5s$	$15,110$	$2.9M_o$
pub	$30s$	$74,510$	$14.3M_o$
movie	$3628s$	$12,423,095$	$2457,6M_o$
SDK2	$335s$	$2,382,720$	$73.2M_o$

compared traces. In practice, we consider $\theta = 0.25$, as threshold of occurrence ratio. One can notice that, for traces of more than $1G_o$, corresponding to approximatively $4,000,000$ events, TED can give a diagnosis in less than $10s$. For the *pub* video of table I, an output is obtained in $0.12s$. The experiments showed that the proposed methods can scale to real application traces.

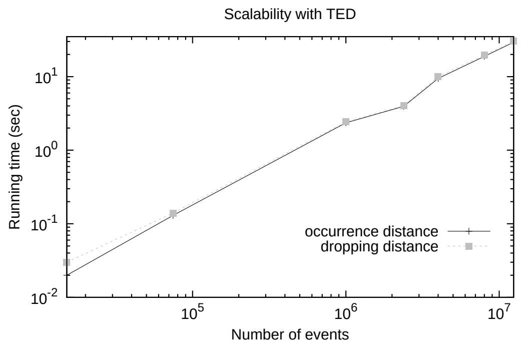


Fig. 4. Running time

Precision: In order to evaluate the accuracy of the diagnosis done by TED, we run TED on a sample of 300 execution traces as shown in Table II. The first observation is that all execution traces initially considered as normal were diagnosed as such by TED. However, the tool gave 14 *false-true* which are execution traces considered by TED as normal but which contain anomalies. Thus, TED has a precision of 95.33%. A reason of this lack of precision can be the value of threshold for *occurrence distance*. We fixed it at $\theta = 0.25$ but it is better to adapt the threshold value to the length of the video decoded. We are currently testing the correlation between the video length and the threshold value.

TABLE II
TED PRECISION

Nb. traces	Initially	With TED
Sample of 300 traces	normal: 130	normal: 144
	abnormal: 170	abnormal: 156

VI. CONCLUSION

To analyse traces of events, programmers use several tools such as trace visualizers ([12]–[15]) and techniques such as tracepoints on the execution traces. These techniques need to have an expert to interpret the graphical representation. In contrast, our work based on distances develops a technique which is a first step towards an automatic anomalies detection. Our approach diagnoses anomalies in an execution trace of multimedia application, by comparison with a reference trace. We use distances as models of comparison and specifically design three distinct distances in order to tackle well-known anomalies of the multimedia domain. We experimentally show the originality of our solution compared to existent distances and show that our proposed approach scales well to real huge application traces. Distances defined in our approach allow to identify a specific problem and give a semantic added-value level to the analysis. Moreover, as all distances, they also provide insights of how far an abnormal trace is from a correct one. We also present a use case on how TED performs the analysis of a trace and conduct some experiments to evaluate TED scalability and accuracy.

We have two research directions. The first direction is to adapt our distances to abstract traces (introduced in [5]) so that our proposal be as generic as possible. The second direction is to enlarge TED to other types of anomalies for instance when the image is completely fuzzy, upside down and/or cut in half. The strength of our contribution is that it is easily extensible to other types of anomalies: for each new anomaly, we only need to follow the same methodology as explained in the paper due to the modularity of TED architecture.

ACKNOWLEDGMENT

This work is supported by French FUI project SoCTrace.

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 24, no. 5, pp. 823–839, 2012.
[2] F. Mörchen, "Time series knowledge mining," 2006.
[3] R. Tavenard, L. Amsaleg, and G. Gravier, "Estimation de similarité entre séquences de descripteurs à l'aide de machines à vecteurs supports," in *Proc. Conf. Base de Données Avancées, Marseille, France*, 2007.
[4] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," in *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*. IEEE, 2000, pp. 39–48.
[5] C. K. Kengne, L. C. Fopa, A. Termier, N. Ibrahim, M.-C. Rousset, T. Washio, and M. Santana, "Efficiently rewriting large multimedia application execution traces with few event sequences," in *KDD Industrial Track (To appear)*, 2013.
[6] Discussion page: Troubleshooting guide. [Online]. Available: http://www.cccp-project.net/wiki/index.php?title=Troubleshooting_Guide
[7] Faq: Play an audio or video file. [Online]. Available: http://windows.microsoft.com/en-us/windows7/play-an-audio-or-video-file-frequently-asked-questions
[8] T. Pang-Ning, M. Steinbach, and V. Kumar, "Introduction to data mining," 2006.
[9] C. Kamdem Kengne, N. Ibrahim, M.-C. Rousset, and M. Tchuent, "Distance-based Trace Diagnosis for Multimedia Applications: Help me TED!" Rapport de recherche, 2013.
[10] H. Mannila and P. Ronkainen, "Similarity of event sequences," in *Proceedings of the 4th International Workshop on Temporal Representation and Reasoning (TIME '97)*, ser. TIME '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 136–.
[11] Gstreamer website. [Online]. Available: http://www.gstreamer.net
[12] B. D. O. Stein, "Pajé trace file format," 2003.
[13] J. Roberts, "Tracevis: an execution trace visualization tool," in *In Proc. MoBS 2005*. Citeseer, 2005.
[14] M. McGavin, T. Wright, and S. Marshall, "Visualisations of execution traces (vet): an interactive plugin-based visualisation tool," in *Proceedings of the 7th Australasian User interface conference - Volume 50*, ser. AUIC '06. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 153–160.
[15] J. Seyster, "Techniques for visualizing software execution," Citeseer, Tech. Rep., 2008.