Réduction à la volée du volume des traces d'exécution pour l'analyse d'applications multimédia de systèmes embarqués

Serge Vladimir Emteu Tchagou

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 6 janvier 2005 - 7 août 2006

Présentée par

Serge Vladimir EMTEU TCHAGOU

Thèse dirigée par **Jean-François Méhaut**
et codirigée par **Luis-Miguel Santana-Ormeno**

préparée au sein **Laboratoire d'informatique de Grenoble**
et de **l'Ecole doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Réduction à la volée du volume des traces d'exécution pour l'analyse d'applications multimédia de systèmes embarqués

Thèse soutenue publiquement le **15 décembre 2015**,
devant le jury composé de :

M. Jérôme Azé
Professeur à l'Université de Montpellier 2, Président
M. Gilles Sassatelli
Directeur de recherche CNRS, Rapporteur
M. Vincent Lemaire
Orange Labs & HDR Paris Sud / Orsay, Rapporteur
Mme Karine Heydemann
Maître de conférence à l'Université Pierre et Marie Curie, Examinateur
M. Jean-François Méhaut
Professeur à l'Université Joseph Fourier, Directeur de thèse
M. Miguel Santana
Directeur du centre SDT à STMicroelectronics, Co-Directeur de thèse
M. Alexandre Termier
Professeur à l'Université de Rennes 1, Co-encadrant de thèse
M. René Quiniou
Chargé de recherche, Co-encadrant de thèse

Chaque difficulté rencontrée doit être l'objet d'un nouveau progrès.
- Pierre de Coubertin.

À mes parents

Remerciements

Nous voici rendu rendu au terme de 4 années de thèse. Ce fut une aventure pleine de rebondissements, de joies mais aussi de peines, tous très enrichissants et desquelles je tire une immense satisfaction. C'est avec une profonde émotion que je manifeste ma reconnaissance vis-à-vis des personnes et entités qui de près ou de loin, m'ont soutenu et permis de franchir cette étape si importante :

Jean-François Méhaut, mon directeur de thèse, pour m'avoir accepté comme doctorant et pour la qualité de son encadrement. Tes questions, conseils et suggestions m'ont toujours permis de m'approprier mon objet de recherche et de mûrir mon raisonnement.

Miguel Santana, mon co-encadreur, pour la confiance et les conseils qui ont structuré nos rapport durant mon séjour au sein de l'entreprise ST ; ainsi que pour les la rigueur professionnelle qui a contribué à façonner le professionnel en devenir que je suis.

Alexandre Termier, mon co-encadreur, pour la patience, la motivation que tu as su me communiquer, mais aussi pour les conseils et la constance des encouragements. À mes yeux, tu es un modèle.

René Quiniou, mon co-encadreur, pour avoir accepté m'accompagner dans la dernière partie de mes travaux de thèse. Un grand merci pour l'intérêt porté à mes travaux, tes observations pointues et ta disponibilité.

Le laboratoire informatique de Grenoble (LIG), STMicroelectronics (ST) et l'IRISA, merci pour l'accueil, le soutien et la collaborations.

Je tiens à remercier tous les membres des équipes de recherches Corse, DREAM, HADAS, SLIDE avec qui j'ai collaboré tout au long de mes travaux.

Je tiens aussi à remercier tous les membres de l'équipe Software Tools and Products Team (SDT) pour l'intégration et collaboration au sein de STMicroelectronics.

Merci à tous les membres de la famille EMTEU à laquelle je suis fière d'appartenir. Merci papa pour la force et la persévérance que tu as su m'inculquer depuis mon enfance. Merci maman, Ulrich, Rostand, Freddy et Rita pour votre soutien et vos encouragements incessants.

Á vous, Christiane (CK), Léon, Juliette, Vanessa, Orléant, Eric, Rodrigue, Innocent, Steeve, Simplice, Blaise, Willibrod, tous mes amis rencontrés durant mon cheminement académique, vous qui avez partagé mon quotidien, m'avez soutenu, grand **MERCI** du fond du coeur.

Résumé

Le marché de l'électronique grand public est dominé par les systèmes embarqués du fait de leur puissance de calcul toujours croissante et des nombreuses fonctionnalités qu'ils proposent. Pour procurer de telles caractéristiques, les architectures des systèmes embarqués sont devenues de plus en plus complexes (pluralité et hétérogénéité des unités de traitements, exécution concurrente des tâches, ...). Cette complexité a fortement influencé leur programmabilité au point où rendre difficile la compréhension de l'exécution d'une application sur ces architectures. L'approche la plus utilisée actuellement pour l'analyse de l'exécution des applications sur les systèmes embarqués est la capture des *traces d'exécution* (séquences d'événements, tels que les appels systèmes ou les changements de contexte, générés pendant l'exécution des applications). Cette approche est utilisée lors des activités de test, débogage ou de profilage des applications. Toutefois, suivant certains cas d'utilisation, les traces d'exécution générées peuvent devenir très volumineuses, de l'ordre de plusieurs centaines de gigaoctets. C'est le cas des tests d'endurance ou encore des tests de validation, qui consistent à tracer l'exécution d'une application sur un système embarqué pendant de longues périodes, allant de plusieurs heures à plusieurs jours. Les outils et méthodes d'analyse de traces d'exécution actuels ne sont pas conçus pour traiter de telles quantités de données.

Nous proposons une approche de réduction du volume de trace enregistrée à travers une analyse à la volée de la trace durant sa capture. Notre approche repose sur les spécificités des applications multimédia, qui sont parmi les plus importantes pour le succès des dispositifs populaires comme les Set-top boxes ou les smartphones. Notre approche a pour but de détecter automatiquement les fragments (périodes) suspectes de l'exécution d'une application afin de n'enregistrer que les parties de la trace correspondant à ces périodes d'activités. L'approche que nous proposons comporte deux étapes : une étape d'apprentissage qui consiste à découvrir les comportements réguliers de l'application à partir de la trace d'exécution, et une étape de détection d'anomalies qui consiste à identifier les comportements déviant des comportements réguliers.

Les nombreuses expériences, réalisées sur des données synthétiques et des données réelles, montrent que notre approche permet d'obtenir une réduction du volume de trace enregistrée d'un ordre de grandeur avec d'excellentes performances de détection des comportements suspects.

Mots-clés : trace d'exécution, application multimédia, analyse à la volée, apprentissage automatique, détection d'anomalies, réduction de données, système embarqué.

Abstract

The consumer electronics market is dominated by embedded systems due to their ever-increasing processing power and the large number of functionnalities they offer. To provide such features, architectures of embedded systems have increased in complexity : they rely on several heterogeneous processing units, and allow concurrent tasks execution. This complexity degrades the programmability of embedded system architectures and makes application execution difficult to understand on such systems. The most used approach for analyzing application execution on embedded systems consists in capturing *execution traces* (event sequences, such as system call invocations or context switch, generated during application execution). This approach is used in application testing, debugging or profiling. However in some use cases, execution traces generated can be very large, up to several hundreds of gigabytes. For example endurance tests, which are tests consisting in tracing execution of an application on an embedded system during long periods, from several hours to several days. Current tools and methods for analyzing execution traces are not designed to handle such amounts of data.

We propose an approach for monitoring an application execution by analyzing traces on the fly in order to reduce the volume of recorded trace. Our approach is based on features of multimedia applications which contribute the most to the success of popular devices such as set-top boxes or smartphones. This approach consists in identifying automatically the suspicious periods of an application execution in order to record only the parts of traces which correspond to these periods. The proposed approach consists of two steps : a learning step which discovers regular behaviors of an application from its execution trace, and an anomaly detection step which identifies behaviors deviating from the regular ones.

The many experiments, performed on synthetic and real-life datasets, show that our approach reduces the trace size by an order of magnitude while maintaining a good performance in detecting suspicious behaviors.

Keywords : execution trace, multimedia application, on-the-fly analysis, machine learning, anomaly detection, data reduction, embedded system.

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

Introduction

Sommaire

Les systèmes embarqués se présentent comme des systèmes électroniques autonomes qui s'appuient sur des logiciels spécialisés afin de fournir des fonctionnalités précises. Depuis quelques années, les systèmes embarqués connaissent une expansion importante à travers les fonctionnalités diverses qu'ils proposent et les différents secteurs d'activités dans lesquels ils sont présents. C'est le cas des smartphones, des récepteurs GPS, des téléviseurs ultra haute définition (UHD) ou des Set-Top Box. Les systèmes embarqués représentent un marché très important selon certains rapports d'analyse du marché de l'embarqué [1, 2, 3]. Outre la domination du marché de l'électronique, ces rapports prévoient un maintien de la croissance du marché de l'embarqué dans les années à venir. Cette croissance est favorisée par la compétition entre les industriels du secteur des semi-conducteurs pour le développement des composants électroniques toujours plus performants et plus fiables avec pour objectif de répondre à :

– la complexité croissante des standards et normes industriels (standard H.265/MPEG-H HEVC, norme HDMI, etc.),
– les besoins constamment grandissants en performance de calcul,
– la minimisation de la consommation d'énergie.

Cette croissance du marché suscite des produits toujours plus complexes en termes d'architecture et de programmabilité, et pour lesquels il est important de réduire les délais de commercialisation. Réduire les délais de commercialisation dans ce secteur, consiste non seulement à optimiser la production de nouveaux produits (systèmes embarqués), mais aussi à faciliter le développement des applications destinées à ces produits. Faciliter le développement d'applications revient à minimiser les délais d'implémentation, de débogage et de validation des applications.

1. ITRS : International Technology Roadmap for Semiconductors, http ://www.itrs.net/papers.html, accédée le 30 juin 2015
2. Forte évolution du marché de l'électronique embarquée sur soi et du marche de l'internet des objets, http ://www.electronique-eci.com/fr/forte-evolution-du-marche-de-l-electronique-embarquee-sur-soi-et-du-marche-de-l-internet-des-objets.html ?cmp_id=7&news_id=2785&vID=8, accédée le 28 juin 2015
3. Chiffres clés : ventes de mobiles et de smartphones, http ://www.zdnet.fr/actualites/chiffres-cles-les-ventes-de-mobiles-et-de-smartphones-39789928.htm, accédée le 30 juin 2015

1.1 Les systèmes embarqués

Pour répondre aux exigences continuellement croissantes des applications et des fonctionnalités, les systèmes embarqués ont essentiellement évolué suivant deux axes : la puissance de calcul et la réduction de la consommation d'énergie. Nous nous intéresserons uniquement à l'aspect puissance de calcul.

Afin de fournir toujours plus de puissance de calcul, les architectures de systèmes embarqués ont évolué des simples circuits de microcontrôleurs vers les architectures complexes constituées des circuits à très large échelle d'intégration. C'est le cas des systèmes sur puce (SoC : System-on-Chip) et les systèmes sur puce à multiprocesseurs (MPSoC : Multiprocessors System-on-Chip). Les SoCs et les MPSoCs sont des circuits intégrés qui contiennent tous les composants d'un ordinateur : un microprocesseur (SoCs) ou plusieurs microprocesseurs (MPSoCs), les composants mémoire, les interfaces d'Entrée/Sortie et d'autres composants électroniques (accélérateurs, processeurs de signal digital (DSPs)). Ces derniers sont dédiés à des traitements spécifiques tels que le décodage audio/vidéo, le traitement de signal ou l'affichage 3D.

À cause de la pluralité et de la diversité de leurs unités de traitements, les systèmes embarqués (SoCs et MPSoCs) proposent une puissance de calcul importante et hétérogène. Pour tirer profit cette puissance de calcul, il est important d'utiliser des modèles appropriés de parallélisation des traitements, ceci afin d'exécuter simultanément plusieurs applications ou plusieurs tâches d'une application. Toutefois, la parallélisation des traitements introduit de nouvelles difficultés et contraintes telles que les difficultés liées aux environnements parallèles (concurrences d'exécution, non-déterminisme à l'exécution, communication inter-tâches, ...) et les contraintes spécifiques aux systèmes embarqués (exécution temps réel, gestion d'énergie, spécificité des composants de traitements, ...). Ces difficultés et contraintes rendent difficile le développement des applications sur les systèmes embarqués. Compte tenu de l'intérêt des consommateurs pour les produits construits à partir de tels systèmes, il est important de développer des applications efficientes et robustes sur ces derniers. Les applications qui présentent des bogues répétitifs ou des comportements inattendus sont très vite critiquées par les consommateurs et peuvent causer la perte d'intérêt, voire la chute des achats pour les produits associés. Afin de faciliter le développement d'applications robustes et efficientes sur leurs produits, les constructeurs des systèmes embarqués proposent aux développeurs des outils de débogage et de profilage adaptés aux architectures des systèmes embarqués. Sur ces systèmes, on retrouve plusieurs types d'applications parmi lesquels les applications multimédia qui font partie des applications les plus sollicitées. Les applications multimédia se déclinent sous plusieurs formes : les applications de décodage audio (lecteurs audio), les applications de décodage audio/vidéo (lecteurs mp4), les applications d'encodage audio/vidéo ou encore les applications de manipulation d'images.

1.2 Le débogage d'applications sur les systèmes embarqués

Le développement d'une application est un processus qui requiert plusieurs étapes dont la conception, l'implémentation, l'analyse de l'application implémentée, la maintenance, etc. L'une des étapes les plus importantes est l'analyse d'applications. Cette étape consiste à vérifier tous les aspects d'une application, notamment les spécifications fonctionnelles (fonctionnalités et résultats attendus) et les spécifications non-fonctionnelles (performances, délais d'exécution, robustesse, fiabilité, etc.). Elle peut être réalisée soit à partir d'une étude minutieuse du code source (ou objet) de l'application - on parle d'analyse statique,

soit à partir des informations dérivées d'une ou plusieurs instances d'exécution de l'application - on parle d'analyse dynamique. Sur les systèmes embarqués, l'analyse dynamique se présente comme la méthode d'analyse des applications la plus appropriée. Elle est moins complexe à réaliser que l'analyse statique et fournit des résultats précis qui intègrent les spécificités des systèmes embarqués.

Sur une plateforme classique, par exemple un PC de bureau, l'analyse dynamique d'une application est réalisée à l'aide des outils de débogage, tels que *GDB*[4], *Valgrind*[5], *WinDbg*[6], ou de profilage tels que *gprof*[7], *perf*[8]. Ces outils fournissent une vue globale du comportement de l'application sur l'architecture d'exécution et permettent de détecter les erreurs de fonctionnement de l'application. Cependant, les outils classiques de débogage et de profilage ne sont pas adaptés à l'analyse des applications sur systèmes embarqués. En effet, face aux contraintes temporelles strictes ou temps réel des applications sur les systèmes embarqués, ces outils introduisent une intrusivité importante qui perturbe le fonctionnement normal de ces applications. C'est le cas des applications multimédia qui présentent des contraintes sur les délais d'exécution de certaines fonctionnalités telles que le traitement d'une trame vidéo dont le délai d'exécution maximal de 40 ms doit être respecté pour une exécution correcte.

Pour contourner ces difficultés, les industriels et les développeurs se sont orientés vers d'autres approches d'analyse dynamique d'applications, les approches basées sur le *traçage*. Le traçage est une technique consistant à collecter et à enregistrer les informations (événements) produites par l'environnement d'exécution au cours de l'exécution d'une application.

Les informations enregistrées à l'issue du traçage d'une application constituent la trace d'exécution de l'application et analyser l'application revient à étudier cette trace d'exécution. De nombreux travaux [PR11, LC13, KIRT13, Ami13, LTP14] ont montré que les traces d'exécution constituent un outil approprié pour le débogage et la validation d'applications sur systèmes embarqués.

Dans les approches usuelles, l'analyse des traces d'exécution est effectuée post-mortem, c'est-à-dire qu'elle est réalisée après la génération et l'enregistrement complet de la trace d'exécution. Ces approches d'analyse sont pour la plupart manuelles. Elles fournissent une représentation graphique du comportement d'une application à partir de sa trace d'exécution. C'est le cas de l'outil *vampir* [KBD+08] ou de l'outil de visualisation de l'application stworkbench [STM] développé par STMicroelectronics.

1.3 Le problème de volumétrie des traces d'exécution pour le débogage

En fonction de la complexité de l'architecture (nombre de composants de traitement observés), du nombre de points de trace et de la durée de traçage, les traces d'exécution peuvent devenir rapidement importantes en taille. Par exemple, la capture et l'enregistrement de la trace d'exécution de l'application DVBTest, application dédiée au décodage et à l'encodage audio/vidéo sur les Set-Top Box de STMicroelectronics, conduisent à environ 150 mégaoctets (Mo) de données pour 1 minute d'exécution, ou encore

4. GDB : The GNU Project Debugger, http ://www.gnu.org/software/gdb/, accédée le 30 juin 2015
5. Valgrind, http ://valgrind.org/, accédée le 30 juin 2015
6. Debugging with GDB under visual studio, http ://wingdb.com/, accédée le 30 juin 2015
7. GNU gprof,https ://sourceware.org/binutils/docs/gprof/, accédée le 30 juin 2015
8. perf : Linux profiling with performance counters, https ://perf.wiki.kernel.org/index.php/Main_Page, accédée le 30 juin 2015

9 gigaoctets (Go) de données pour 1 heure d'exécution. De même, pour certains cas d'utilisation au sein de STMicroelectronics tels que les tests de validation ou d'endurance au cours desquels une application est testée pendant plusieurs heures, voire plusieurs jours, les traces d'exécution peuvent atteindre plusieurs téraoctets. L'analyse de telles quantités de traces est très compliquée. Pour illustrer cette difficulté, nous présentons à la Figure 1.1 une approche d'analyse des communications inter-threads d'une application capturées dans un trace de quelques millisecondes d'exécution. La quantité d'informations représentant les communications inter-threads constitue une difficulté pour l'analyse et la compréhension de l'application étudiée.

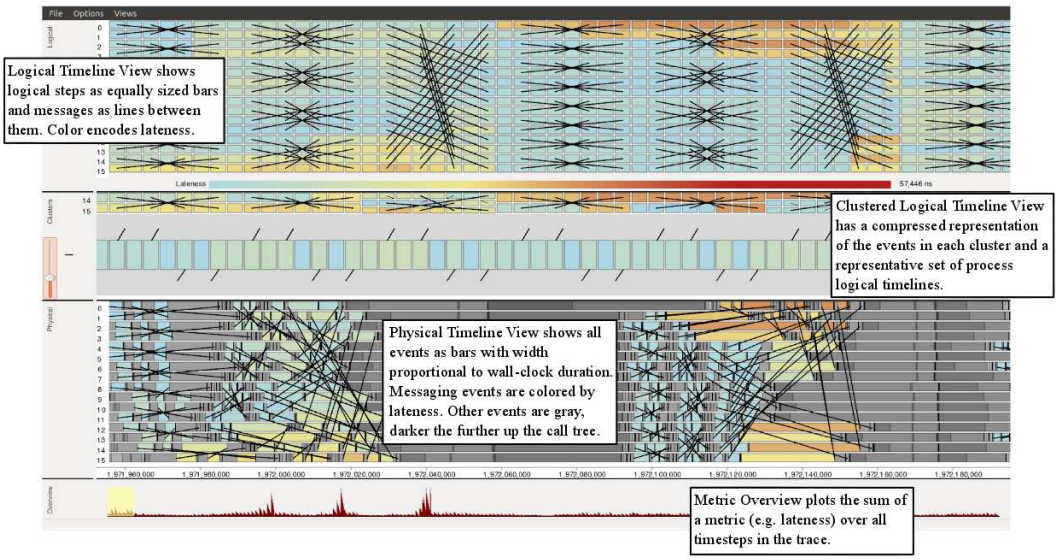


FIGURE 1.1 – Visualisation des communications inter-threads à partir de l'outil de visualisation de trace, *ravel* [IBJ+14].

Face aux difficultés des approches manuelles, des approches automatiques d'analyse post-mortem ont été proposées. Ces approches reposent sur des techniques de comparaison [KIRT13], de fouille de données [LTP14, CBT+12]. Elles consistent généralement à détecter automatiquement des problèmes tels que les ruptures de périodicité dans le comportement des applications, les problèmes de contention ou les fautes dans l'exécution des applications.

Face aux problématiques liées au volume important de données dans les traces d'exécution, une solution serait de réduire le volume des traces d'exécution de manière à ne conserver dans les traces que les éléments pertinents pour l'analyse de l'application.

1.4 Contributions

Réduire le volume d'une trace d'exécution consiste à effectuer un prétraitement sur la trace d'exécution afin de résumer ou de diminuer la quantité de données contenue dans la trace tout en conservant les informations importantes contenues dans les données d'origine. Dans cette thèse, nous proposons une nouvelle approche de réduction des traces basée sur l'analyse à la volée des traces d'exécution. Nous partons de

l'observation que les traces d'exécution des applications, particulièrement celles des applications multimédia, sont constituées en majorité de données qui se répètent avec des variations plus ou moins importantes et qui représentent les périodes de fonctionnement normal. Ces données répétitives ne présentent aucun intérêt lorsque l'application tracée s'exécute correctement. Etant donné que le débogage d'une application porte principalement sur l'analyse des périodes de fonctionnement problématique de cette application, il serait intéressant de ne stocker que les données qui apparaissent pendant ces périodes de fonctionnement problématique (Figure 1.2).

FIGURE 1.2 – Illustration des régimes de fonctionnement répétitifs et problématiques présents dans la trace d'exécution d'une application.

Par conséquent, nous souhaitons réduire le volume de trace d'exécution au fur et à mesure de leur génération afin de ne retenir que les portions de traces qui correspondent aux périodes d'exécution problématique de l'application analysée, et ce, sans connaissance a priori sur cette application.

Nos contributions se résument ainsi :

– Analyser à la volée les traces d'exécution tout en étant capable de suivre leur débit de génération. Nous proposons une approche d'analyse des traces d'exécution au fur et à mesure qu'elles sont générées, sans besoin qu'elles soient stockées au préalable.
– Apprendre automatiquement des régimes de fonctionnement réguliers (normaux) d'une application à partir de sa trace d'exécution. Nous proposons un modèle d'identification des régimes réguliers sans connaissance à priori, à partir des distributions d'événements et de la redondance dans la trace.
– Utiliser une approche de détection d'anomalies de l'état de l'art pour identifier des régimes de fonctionnement problématique d'une application. Nous proposons un modèle de représentation des distributions d'événements d'une trace d'exécution dans un espace multi-dimensionnel, où chaque dimension représente un type-événement, et à partir duquel nous identifierons les distributions d'événements qui dévient des distributions associées aux régimes réguliers de l'application.
– Réduire la trace d'exécution pour n'en retenir que les portions liées aux comportements problématiques des applications au cours de leur exécution. Nous proposons de retenir uniquement en résultat de notre approche, les portions de la trace générée dont les distributions d'événements diffèrent de celles associées aux régimes réguliers.

1.5 Le contexte scientifique

Cette thèse a été financée par un contrat CIFRE ANRT dans le cadre d'une collaboration entre STMicroelectronics et le Laboratoire Informatique de Grenoble (LIG) de l'Université de Grenoble Alpes. Au sein de STMicroelectronics, les travaux ont été réalisés dans l'équipe SDT [9] dont le rôle est de fournir aux clients les outils appropriés pour l'observation et le débogage des applications multimédia sur les systèmes embarqués développés par STMicroelectronics. Ce travail de thèse s'inscrit ainsi dans l'élaboration des outils de débogage pour les systèmes embarqués.

La partie scientifique et académique de cette thèse a été effectuée dans différentes équipes de recherche :

– l'équipe Corse dont les champs de recherche portent sur les problématiques d'optimisation des compilateurs et systèmes d'exécution pour les environnements de calculs à hautes performances (HPC) et les systèmes embarqués ;
– les équipes DREAM et SLIDE dont les champs de recherche portent sur les problématiques de fouille de données et d'extraction de connaissances dans de grandes quantités de données.

1.6 L'organisation de la thèse

Le reste de ce document est organisé de la manière suivante :

– Le Chapitre 2 présente le contexte de la thèse en donnant une vue globale sur les architectures des systèmes embarqués et les applications multimédia. Il présente les traces d'exécution comme principal outil d'analyse (débogage et profilage) des applications sur les systèmes embarqués et les problématiques de volume des traces d'exécution.

9. SDT : Software Tool Products team

- Le Chapitre 3 présente les approches de réduction et de détection d'anomalies sur les traces d'événements.
- le Chapitre 4 présente notre approche de réduction à la volée des traces d'exécution à partir de la détection d'anomalies.
- le Chapitre 5 présentent les expérimentations réalisées en vue d'évaluer notre approche de réduction des traces d'exécution.
- La conclusion et les perspectives sont présentées au Chapitre 6.

CHAPITRE 2

Background : Systèmes embarqués et analyse des traces d'exécution

Sommaire

Ce chapitre présente le contexte de la thèse qui porte sur l'étude des applications pour les systèmes embarqués.

Nous commencerons par présenter les architectures des systèmes embarqués les plus utilisés de nos jours. Par la suite, nous exposerons les difficultés liées au débogage, à l'optimisation et à la validation des applications sur les systèmes embarqués. Puis, nous présenterons le concept de trace d'exécution ainsi que son importance pour l'analyse des applications dans le contexte de l'embarqué. Nous montrerons également les limites actuelles de l'utilisation des traces d'exécution pour l'analyse des applications, notamment le volume important de données.

2.1 Les systèmes embarqués

Au début des années 60, les premiers systèmes embarqués étaient uniquement des *micro-contrôleurs*. Leur architecture reposait sur un circuit intégré constitué d'un unique CPU, d'une mémoire et de périphériques d'Entrée/Sortie dédiés [Aug83]. Ils étaient conçus pour répondre, à faible coût, aux besoins en calcul de certaines tâches prédéfinies. À cet effet, ils étaient utilisés pour des applications orientées contrôle, par exemple les contrôleurs des feux de circulation, les contrôleurs d'ascenseurs ou de machines à laver.

Cependant, avec l'évolution des applications en terme de complexité d'une part et d'autre part les exigences du portage des applications des micro-ordinateurs vers l'embarqué, les micro-contrôleurs sont vite devenus inadaptés. La capacité de calcul fournie par ces micro-contrôleurs est devenue insuffisante face aux exigences en performance des applications.

Ces exigences, sans cesse croissantes en performance de calculs, ont été fortement favorisées par un contexte concurrentiel du marché des applications. Ce contexte, motivé par un important courant d'innovation, a influencé ainsi le développement des systèmes embarquées sur plusieurs axes [WJM08]. Parmi

ceux-ci, *l'influence portée par les standards* et *l'influence portée par la complexité des applications* sont les plus notables.

L'influence portée par les standards se caractérise par la volonté des industriels d'orienter le design des systèmes embarqués vers les standards des applications. L'objectif visé est d'adapter les architectures d'exécution aux exigences des applications. Les standards génèrent un marché conséquent pour les systèmes embarqués, mais par contre, imposent des défis importants liés aux contraintes des applications, mais aussi aux contraintes des architectures d'exécution (haute précision, timings fixes de traitement, faible consommation d'énergie). Une solution de satisfaction de ces contraintes est l'implémentation matérielle des fonctionnalités associés à ces standards. C'est le cas du standard MP3 (MPEG-2 Audio Layer III) pour lequel un amplificateur audio et un processeur spécialisé dans la conversion du numérique/analogique pour le décodage des fichier MP3 sont intégrés sur les baladeurs numériques MP3 ; ou encore du standard H.264 qui est implémenté sur une set-Top box par un décodeur Dual HD et un encodeur H.264 respectivement pour le décodage et l'encodage des données vidéo au format H.264.

L'influence portée par la complexité des applications repose sur le fait que la plupart des applications destinées aux systèmes embarqués sont des systèmes de plusieurs algorithmes. Ce qui occasionne des différences de comportements liées aux tâches de l'application. C'est le cas des applications d'encodage MPEG-2 [Has97] qui effectuent globalement deux tâches principales : l'estimation du mouvement et la transformée en cosinus discrète (DCT). Ces tâches présentent des profils de calculs différents. En effet, l'estimation de mouvement exhibe un profil où les calculs sont dominés par les accès mémoire pendant que la transformée en cosinus directe présente un profil caractérisé par des calculs complexes. Pour prendre en compte cette hétérogénéité des traitements, les industriels équipent les systèmes embarqués de processeurs de calculs spécialisés et adaptés à différents profils de traitement.

Suite à ces influences, les systèmes embarqués ont vu leur architecture évoluer principalement suivant l'aspect puissance de calcul. Les premiers systèmes embarqués (micro-contrôleurs) disposaient au départ d'un micro-processeur 8 bits. Dans l'optique de fournir plus de performance, ils ont évolué vers les versions 16-bits puis 32 bits (ARM, MIPS), proposant suffisamment de puissance de calcul pour assurer le traitement d'applications de décodage H.264, routage réseau et téléphonie mobile. Par la suite, la puissance des systèmes embarqués s'est orientée vers la pluralité des unités de calcul, donnant naissance aux architectures System-on-Chip (SoCs) et MultiProcessor System-on-Chip (MPSoCs) [BR03].

SoCs

Les architectures System-on-Chip (SoC) constituent l'une des évolutions majeures des systèmes embarqués. Elles équipent la plupart des équipements embarqués actuels (set-top box, smart-phones, consoles de jeux). Une architecture SoC est un circuit intégré qui contient, à l'intérieur d'une seule puce, tous les composants d'un ordinateur minimal : un processeur, une mémoire, des interfaces d'Entrée/Sortie. En plus de ces composants, et suivant les domaines d'application, on y retrouve des systèmes électroniques (Figure 2.1) tels que des supports de traitement de signal (digital, analogique ou électromagnétique), des accéléromètres ou des capteurs de température. L'objectif consiste à diminuer le nombre de composants électroniques distincts et à en regrouper le plus possible sur une seule puce.

L'émergence des SoCs a été fortement encouragée par plusieurs facteurs dont la réduction de la consom-

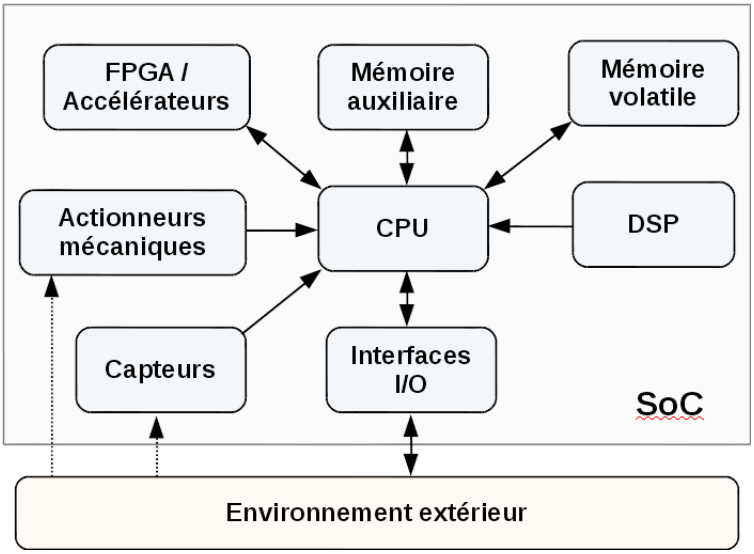


FIGURE 2.1 – Schéma typique d'un système embarqué

mation électrique, l'augmentation de la performance de traitement nécessaire aux applications et la baisse des coûts motivée par la croissance du marché de l'électronique mobile (smartphones, baladeurs MP3, récepteurs GPS).

Un exemple intéressant de système embarqué intégrant une architecture SoC est la set-top box. Une set-top box (ou décodeur TV) est un adaptateur qui transforme un signal externe (analogique ou numérique) en un contenu affichable sur un écran de télévision. Les fonctionnalités des set-top box se sont considérablement diversifiées avec l'implantation de la télévision numérique. Cette diversification se caractérise par l'ajout des fonctionnalités de décodage et d'encodage du signal numérique. Elle a été largement favorisée dans les années 1990 par l'élaboration de la norme MPEG-2 [1], l'une des premières normes à définir les spécifications de compression d'image et de son dans le but d'améliorer le stockage et la transmission de la vidéo à partir du signal numérique [Tud95]. La board STi5107 (Figure 2.2) est l'une des premières set-top box à fournir la puissance de calcul nécessaire pour le support de la norme MPEG-2. Elle a été développée par l'entreprise ST MicroElectronics.

Cette board contient un SoC constitué principalement :

- d'un processeur *ST20*, cadencé à 200MHz, responsable du traitement des applications, de l'orchestration du décodage MPEG-2 à travers le démultiplexage des données et la synchronisation des tâches de décodage audio/vidéo,
- d'un décodeur video (MP@ML video decoder) chargé d'extraire des images d'un flux d'image compressé suivant la norme MPEG-2,
- d'un décodeur audio (Audio decoder) responsable du traitement du flux audio compressé,
- des périphériques d'acquisition de signal vidéo par câble, satellite ou terrestre, et des périphériques de sortie audio/vidéo.

Les architectures SoC sont qualifiées d'architectures hétérogènes car elles disposent de différentes unités de traitement associées à des jeux d'instructions différents et parfois à une programmabilité différente.

1. standard MPEG-2 : http ://mpeg.chiariglione.org/standards/mpeg-2

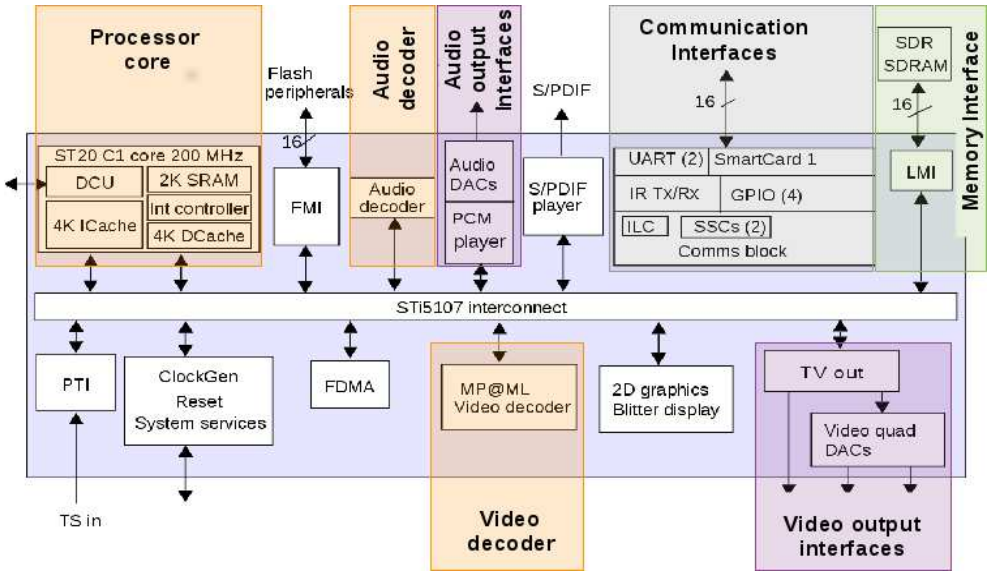


FIGURE 2.2 – L'architecture SoC de la board STi5107

Cette hétérogénéité liée à la programmabilité des unités de traitement rend difficile la mise au point des applications sur les architectures SoC.

MPSoCs

Pour répondre à l'apparition de nouveaux services qui nécessitent des calculs importants, les architectures des systèmes embarquées, à l'instar des set-Top box, ont vu leur nombre de microprocesseurs se multiplier. À cet effet, les set-Top box ont évolué des architectures SoC vers les architectures MPSoC (MultiProcessor-on(-a)-Chip). C'est le cas de la set-top box Tv Home Gateway [2] qui est équipée d'un MPSoC doté d'un processeur dual-core ARM.

De manière générale, l'évolution des SoCs en terme de puissance de calcul s'est traduite par une augmentation des unités de traitement (ou cœurs) au sein du microprocesseur, ce qui a donné naissance aux architectures MPSoCs. Les coeurs des MPSoCs sont reliés entre eux par un réseau de communication (Network-on-Chip ou *NoC*) [TBS+11].

L'orientation des microprocesseurs des systèmes embarqués vers le multicœur a entraîné l'exploitation d'une autre forme de puissance de calcul qu'est le parallélisme. Par ailleurs, les coeurs des microprocesseurs des MPSoCs consomment très peu d'énergie, ce qui permet aux MPSoCs d'offrir un meilleur compromis entre la puissance de calcul et la consommation d'énergie, en comparaison aux architectures multicœurs conventionnelles.

Les architectures MPSoCs se retrouvent de plus en plus dans les équipements embarqués de dernières générations. C'est le cas du MPSoC Snapdragon 801 Krait 450 [3] qui équipe le Samsung Galaxy S5. On peut aussi citer le MPSoC *PNX85500* qui intègre un microprocesseur multi-coeurs MIPS32 24KE[TAB+05] destinés aux écrans TV LCD.

2. STiH416 description : http ://www.st.com/web/catalog/mmc/FM131/SC999/SS1633/PF253155
3. Snapdragon 801 specifications : https ://www.qualcomm.com/products/snapdragon/processors/801

Sur la Figure 2.3, nous présentons l'architecture du MPSoC *Snapdragon 801* qui équipe le smartphone Samsung S5. Ce MPSoC est équipé d'un processeur quad-core (Qualcomm Krait 400) cadencé à 2.5 GHz, d'un GPU (Qualcomm Adreno 330), d'un processeur de signal digital (Hexagon V50), d'un modem 4G LTE, d'un processeur de lecture et capture vidéo Ultra HD, d'un moteur de gestion efficace des capteurs et des modules de gestion de la connectivité (Bluetooth, Wifi et USB), de la localisation (GPS), de la camera et de l'affichage haute résolution.

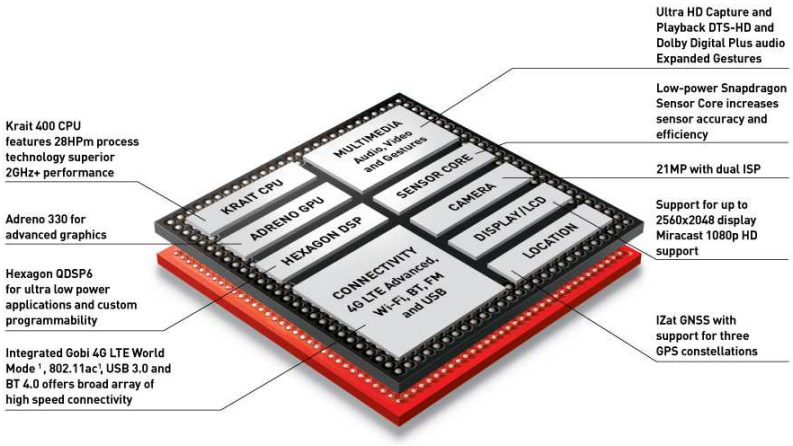


FIGURE 2.3 – L'architecture MPSoC Qualcomm Snapdragon 801 [Qua15]

Les MPSoCs ont été conçus pour héberger les applications qui nécessitent des traitements complexes, par exemple les applications multimédia. Cependant, la structure et la pluralité des unités de traitement que présentent ces architectures amplifient les difficultés liées à la parallélisation des applications (concurrence d'exécution, non déterminisme lié à l'exécution, communications inter-tâches) et l'hétérogénéité de la programmabilité.

Dans la suite de ce chapitre, nous nous intéresserons à l'un des types des applications les plus répandues sur les systèmes embarqués : les applications multimédia. Nous présenterons l'architecture des applications multimédia, puis nous parlerons de la complexité liée à ces applications. Nous terminerons par les méthodes et techniques de validation qu'imposent le développement des applications sur les systèmes embarqués.

2.2 Les applications multimédia

Le multimédia se définit comme une combinaison de différentes formes d'information : texte, image, vidéo ou audio accessibles à travers différents formats. Ces différentes formes d'information sont manipulables grâce à des applications dites applications multimédia.

Le multimédia se présente comme un domaine important d'innovation et joue un rôle essentiel sur le marché des systèmes embarqués. Il ne cesse d'évoluer en terme d'applications telles que la vidéo 3D, la visio-conférence, mais aussi en terme de technologies telles que le téléviseur 4K, le projet Oculus Rift qui porte sur l'élaboration des casques pour la réalité virtuelle, ou le projet Microsoft Hololens qui porte sur l'élaboration des casques pour la réalité augmentée.

La structure des applications multimédia repose esssentiellement sur trois couches (Figure 2.4) : la couche application, la couche intergicielle ou framework multimédia et le système d'exploitation.

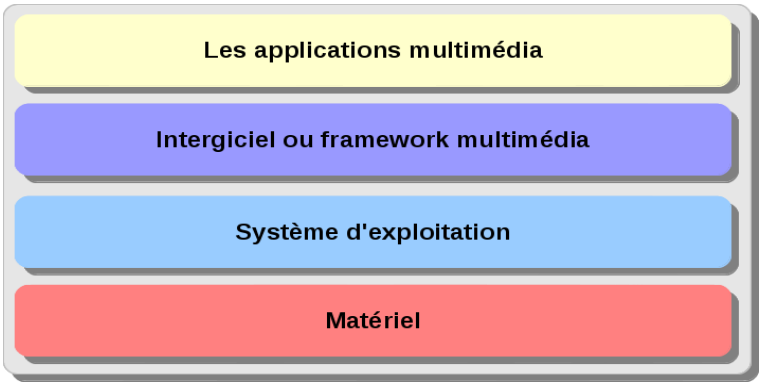


FIGURE 2.4 – L'infrastructure multimédia

Le système d'exploitation est la couche qui sert d'interface de communication entre la couche intergicielle et la couche matérielle. La couche intergicielle se présente comme la composante essentielle de l'infrastructure multimédia. Elle définit et fournit les services de gestion des données multimédia. Par ailleurs, elle assure l'indépendance des applications multimédia de la plate-forme d'exécution. La couche application sert d'interface d'accès aux données multimédia et rend possible la transmission des commandes de traitements à réaliser sur les données multimédia.

Les fonctionnalités de la couche intergicielle sont généralement réalisées par les frameworks multimédia tels Gstreamer [4] ou VLC [5]. Ces frameworks offrent une variété de composants logiciels qui peuvent être combinés par un pipeline afin de réaliser une fonctionnalité précise. La structure du pipeline dépend généralement du type d'application multimédia. La Figure 2.5 présente un exemple de pipeline de composants dédiés au décodage audio/vidéo.

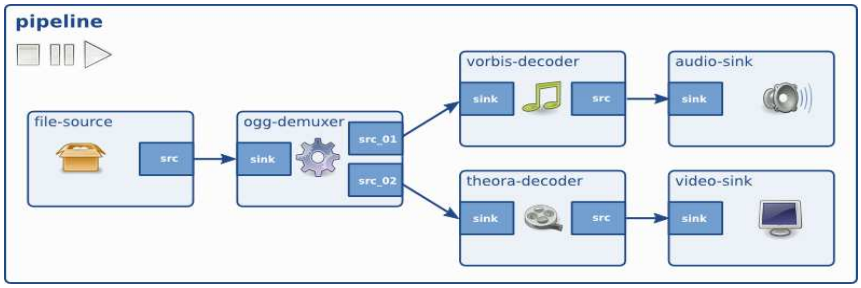


FIGURE 2.5 – Application multimédia reposant sur un pipeline GStreamer [GSt15]

À travers leur structure modulaire, les frameworks multimédia facilitent la mise en œuvre des applications multimédia. Le pipeline qu'implémentent les frameworks multimédia induit un modèle de programmabilité bien adapté au parallélisme et à l'hétérogénéité des architectures MPSoCs [POH09].

Les applications multimédia ont besoin d'une puissance importante de calcul pour réaliser efficacement leurs fonctionnalités. Pour acquérir cette puissance de calcul sur les MPSoCs, les frameworks multimédia ont vu leur modèle de programmation passer du paradigme impératif vers le modèle dataflow [KS05]

4. Framework multimédia Open Source : http ://gstreamer.freedesktop.org/
5. VLC media player : http ://www.videolan.org/vlc/

[BMR10] ou encore le modèle orienté composant [BDRL05]. En effet, les modèles dataflow et orientés composants permettent d'extraire, sur les MPSoCs, un parallélisme à moyen grain. Ce type de parallélisme à l'exemple du SIMD (Single instruction Multiple Data) est bien adapté pour le traitement des flux de données audio et vidéo à travers des opérations vectorielles.

Cependant, la programmabilité des architectures MPSoC reste difficile. À cela s'ajoute les problématiques liées au développement des applications multimédia. Les applications multimédia reposent sur des fonctionnalités complexes en terme de traitement (décodage, encodage, compression). Leurs implémentations nécessitent la collaboration de plusieurs développeurs, conduisant ainsi à une connaissance partielle de l'application au niveau de chaque développeur, voire de chaque équipe. Par ailleurs, le développement des applications de qualité sur ces architectures impose d'importantes phases de débogage, de tests et de validation. Ces phases font partie de l'analyse d'applications qui est une étape importante dans le processus de développement de ces dernières et nécessite l'utilisation des méthodes et techniques appropriées pour l'analyse.

2.3 Méthodes d'analyse des applications

L'analyse d'une application peut se définir comme l'ensemble des processus et techniques qui permettent d'étudier automatiquement tous les aspects d'une application à savoir le code, les fonctionnalités et le comportement. Elle se concentre non seulement sur la conformité d'une application par rapport aux spécifications fonctionnelles (fonctionnalités attendues de l'application) mais aussi par rapport aux spécifications non fonctionnelles (performance, rapidité d'exécution, fiabilité, ...). L'analyse des applications fait partie du processus de développement d'une application et représente à elle seule environ 60% de ce processus[Mur12].

L'analyse des applications peut se faire soit à travers l'analyse statique, soit à travers l'analyse dynamique. L'analyse statique d'une application consiste à étudier le code de l'application afin de dériver des informations sur son exécution sans pour autant y procéder. Elle se fait uniquement à partir d'une lecture et analyse du code source de l'application, ou de son code objet dans certains cas. Elle permet de détecter des erreurs qui vont du non respect de la syntaxe du langage d'implémentation, au non respect des spécifications fonctionnelles et non fonctionnelles. Elle propose des techniques qui reposent sur des modèles mathématiques construits à partir des propriétés formelles et fonctionnelles des applications. Ces techniques se regroupent en 3 grandes catégories : *les techniques basées sur les preuves formelles* (PVS [ORS92], COQ [BC04], ACL [KMM00]), *les techniques basées sur la modélisation abstraite* [Pnu77, CES86, BHJM09] et *les techniques basées sur l'approximation sémantique*[CC77].

L'analyse statique présente des difficultés dans le contexte des systèmes embarqués du fait qu'elle s'appuie non pas sur le système réel, mais plutôt sur un modèle approché du système réel. Ainsi, ce caractère approximatif du modèle donne lieu à des résultats qui peuvent être imprécis ou contenir un grand nombre d'erreurs. Cette imprécision s'accroît suivant la pluralité et l'hétérogénéité des architectures d'exécution telles que les MPSoCs. Les solutions proposées pour résoudre le problème de précision induisent des modèles complexes et coûteux à analyser, et cela quelque soit la méthode utilisée (preuves formelles, model-checking ou interprétation abstraite).

L'analyse dynamique quant à elle repose sur des évaluations du code et du comportement d'une application grâce à des exécutions. L'idée générale consiste à étudier les données et résultats obtenus à partir d'une ou plusieurs exécutions d'une application sur ces données. Son but réside dans la compréhension du comportement dynamique des applications d'une part, et d'autre part dans la détection et la résolution des erreurs ou vulnérabilités des applications. Ces erreurs proviennent généralement des causes subtiles difficilement détectables à l'analyse statique. Il existe plusieurs techniques d'analyse dynamique :

- *les tests* qui sont une série d'activités permettant d'examiner de manière partielle ou complète les fonctionnalités des applications [Mye04].
- *les assertions* qui reposent sur la construction des prédicats à vérifier en vue d'une exécution cohérente du programme testé [Hoa69].
- *le débogage interactif* qui repose sur l'utilisation d'un outil (débogueur) pour explorer, analyser et comprendre l'exécution d'une application [Pou14].
- *la couverture de code* qui utilise une métrique pour décrire le degré et la qualité de test du code source [MM63, MSB11].
- *l'instrumentation de code* qui regroupe l'ensemble des techniques basées sur l'ajout des instructions spéciales dans le code. Ces instructions permettent de générer des informations sur le comportement des applications pendant leur exécution. Elles sont appelées *points de trace* [KWK10].

Les techniques d'analyse dynamique sont complémentaires et peuvent s'appliquer les unes après les autres pendant la phase d'analyse d'une application. Cependant, la technique la plus pertinente est l'instrumentation de code. En effet, elle fournit une observation plus ou moins fine du comportement d'une application et cela de manière interactive ou non. Les outils usuels liés à cette technique sont pour la plupart des cas les outils de débogage (GDB, Intel Debugger) ou de profilage (Valgrind, Gprof). Toutefois, ces outils présentent un inconvénient considérable : leur forte intrusivité. Cette forte intruisivité donne lieu à des problèmes de modification des délais d'exécution des fonctionnalités de certaines applications, notamment les applications multimédia pour lesquelles les contraintes temporelles d'exécution sont fortes. Par ailleurs, ces outils sont inadaptés sur les MPSoCs car ils ne prennent pas en compte les accélérateurs ou les processeurs de signal (Digital Signal Processor ou DSP) présents sur ces architectures. De ce fait, la composante de l'instrumentation de code qui sied le mieux à l'analyse des applications sur les systèmes embarqués est la trace d'exécution. Une trace d'exécution se présente comme une suite d'événements produits par un système pendant l'exécution d'une application.

L'analyse d'applications à partir des traces d'exécution passe par une étape préalable, appelée le traçage. Cette étape consiste à observer l'exécution d'une application sur une architecture d'exécution. Elle est réalisée à partir des composants de traçage qui peuvent être matériels ou logiciels. Les outils de traçage logiciels reposent principalement sur les bibliothèques spéciales ou les modules du noyau. C'est le cas de LTT [6] pour le système d'exploitation Linux ou la bibliothèque KPTrace [7] pour STLinux [8], ou encore de l'application EMBera [PR11] qui est une approche à base de composants pour l'observation des systèmes multiprocesseurs. Ces outils sont implémentés de manière à minimiser considérablement l'intrusivité induit par le traçage lors de l'exécution de l'application. Cependant, ils peuvent entraîner des latences importantes

6. kit de traçage Linux - next generation. https ://www.lttng.org/, accédée le 06 mai 2015
7. KPTrace - une solution novateur de traçage pour STLinux - http ://www.stlinux.com/devel/traceprofile/kptrace, accédée le 06 mai 2015
8. Système d'exploitation construit par ST MicroElectronics, basé linux, destinés aux architectures embarquées construits sur les processeurs ARM ou ST

pour la gestion des données contenues dans la trace. Les données des traces peuvent devenir très importantes en terme de taille au point de ne plus tenir en mémoire, entraînant ainsi la mobilisation du processeur pour leur gestion. Afin de décharger le microprocesseur des systèmes embarqués de la gestion des données de trace, les industriels ont associé aux solutions logicielles de traçage, des solutions matérielles à travers la mise en place des composants matériels que sont l'ETM (Embedded Trace Macrocell), le PTM (Program Trace Macrocell) et la sonde JTAG [9] (cf. Figure 2.6) présents sur l'architecture des systèmes embarqués et qui permettent de réaliser la collecte et le transport les données des traces.

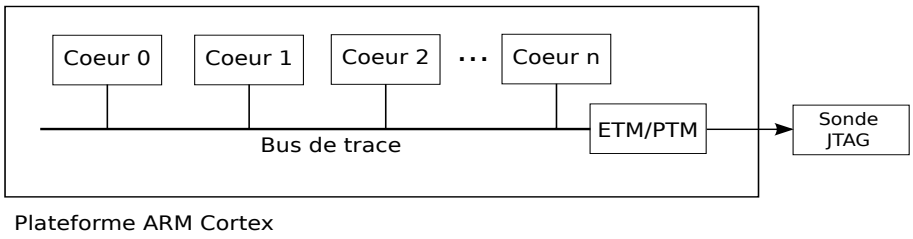


FIGURE 2.6 – Composants de traçage ETM/PTM sur une plateforme multi-cœur ARM [PR11]

De manière générale, le traçage sur les systèmes embarqués débute par la sélection des entités à observer. Ce sont les unités de calculs (processeurs, accélérateurs), les flots d'exécution, les interruptions, les symboles du noyau (changements de contexte, appels systèmes ou fonction utilisateurs et arguments associés) ou encore les opérations sur la mémoire (allocation, libération). Par la suite, la collecte des données, pendant l'exécution d'une application, est effectuée à partir de la lecture des registres des processeurs ou des registres mémoires. Ces données sont transportées sur le bus de trace vers les composants ETM et PTM qui fournissent le support d'estampillage de transport vers la sonde JTAG.

La trace d'exécution se présente donc comme la meilleure solution d'analyse des applications sur les systèmes embarqués. Un exemple de trace est présenté à la Figure 2.7. Chaque ligne de trace constitue un événement produit sur le système étudié.

De nombreuses méthodes d'analyse de trace d'exécution ont été élaborées pour permettre l'analyse des applications sur les systèmes embarqués.

2.4 Analyse des traces d'exécution des applications embarquées : cas des applications multimédia

Les traces d'exécution se définissent formellement comme une séquence d'événements horodatés qui décrivent les différentes activités d'une application pendant son exécution. Ces activités peuvent être décrites à plusieurs niveaux, du niveau matériel (compteurs d'instruction, défauts de caches) au niveau applicatif (appels systèmes, invocation d'une fonction utilisateur, changement de contexte de processus).

Les méthodes d'analyse des traces d'exécution reposent sur l'analyse des données des traces d'exécution dans le but de diagnostiquer des problèmes, vérifier des propriétés ou encore optimiser les fonctionnalités

9. JTAG - nom de la norme IEEE 1149.1 intitulé "Standard Test Access Port and Boundary-Scan Architecture"

```
52130095   context_switch  20 951
52130101   enter_function  4 0x76e0 4096 0x00
52130132   sys_call  0x76e0f000
52130138   context_switch  951 20
52130144   context_swtich  20 0
52130350   interrupt  179
52130356   end_interrupt
52130359   interrupt  142
52130438   end_intterupt
52130444   context_switch  0 20
52130461   context_swtich  20 951
52130467   exit_function  4 0x76e0 4096 0x0
52130497   enter_function  0x6eac 0 0xf000 632 0x00
52130504   sys_call 0x400b1e1c
52130505   context_switch  951 20
52130518   context_switch  20 0
```

FIGURE 2.7 – Exemple de trace

des applications. Ces méthodes sont généralement réalisées post-mortem, c'est à dire qu'elles sont faites à la fin de la collection des données de trace. À cet effet, les données des traces d'exécution sont stockées au préalable dans un fichier ou dans une base de données.

La plupart des outils d'analyse de trace reposent sur les techniques de visualisation [CZH+08]. À partir des modèles construits sur les données, ces outils proposent diverses représentation des fonctionnalités de l'environnement tracé, notamment le profilage CPU ou mémoire, le chronogramme d'exécution des tâches, etc. Dans le cas du chronogramme d'exécution des tâches, certaines techniques exploitent le diagramme de Gantt [Wil03] pour représenter le séquencement des tâches dans le temps. C'est le cas de l'outil d'analyse proposé par ST MicroElecronics, STLinux Trace Viewer dont une illustration est donnée à la Figure 2.8.

Pour faciliter l'exploration des événements d'une trace d'exécution, l'outil STLinux Trace Viewer décompose la trace d'exécution en pages où une page contient un sous-ensemble d'événements contigus. La Figure 2.9 représente une page de trace d'exécution dont la durée associée à la page est de 688.636 ms ($\simeq 0.7$ s) pour une trace complète d'une durée de 71 s d'exécution, équivalent au total à 99 pages.

Prenant en compte le fait qu'une analyse plus détaillée peut nécessiter un ou plusieurs zooms sur une page (Figure 2.8), l'analyse sur une ou plusieurs pages peut devenir facilement fastidieuse et complexe, voire impossible sur certains volumes de trace telles que celles générées suite à un test effectué sur plusieurs heures voire jours(*test d'endurance*). En outre, sur la Figure 2.9 sont représentées uniquement les activités exécutées par un seul coeur du processeur multicoeurs. Les architectures MPSoCs possèdent des microprocesseurs multicœurs, les traces d'exécution produites par ces architectures sont encore plus denses et plus volumineuses en terme d'événements. Confrontées à l'énorme volumétrie de données, les techniques de visualisation deviennent très vite difficiles à exploiter pour analyser les applications sur les architectures MPSoCs.

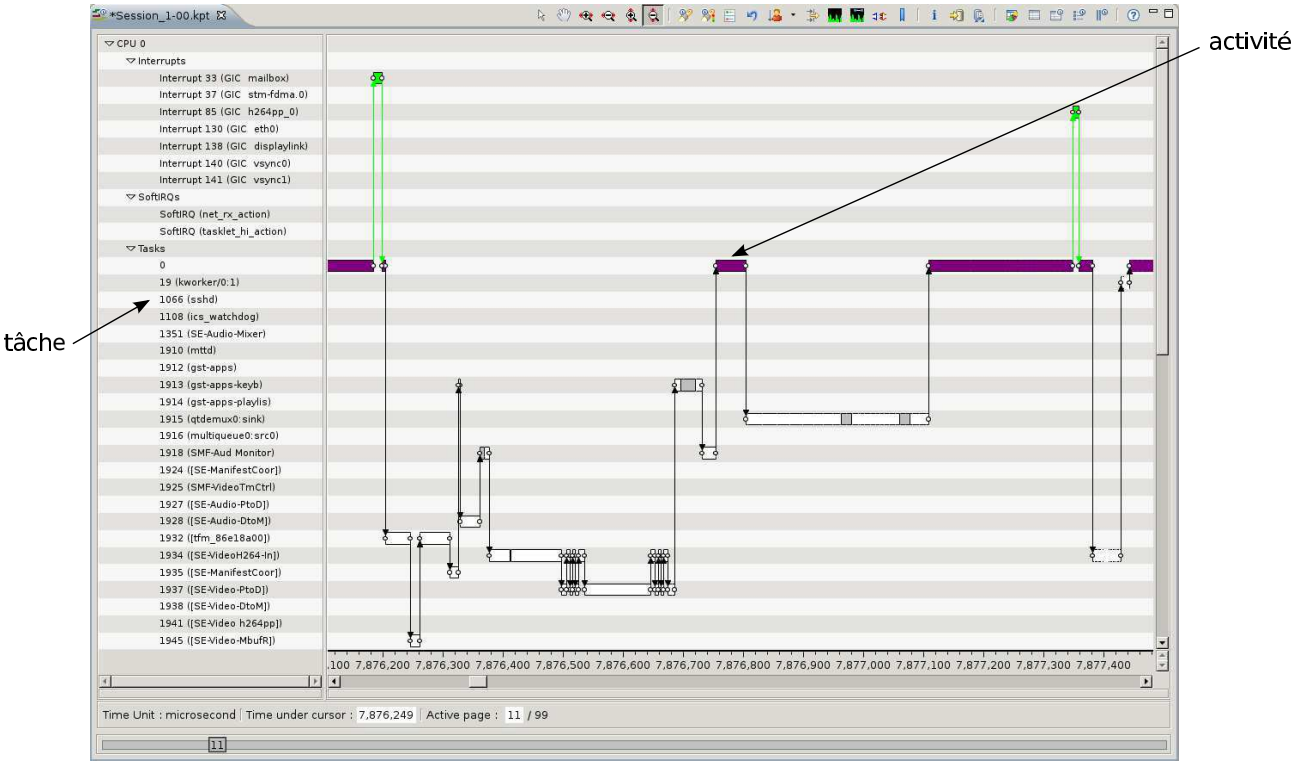


FIGURE 2.8 – Zoom sur une page d'événements sur l'outil STLinux Trace Viewer.
Représentation des événements capturés par l'environnement KPTrace sur une board STiH416 pendant l'exécution d'une application multimédia. Les différentes tâches sont représentées sur la partie gauche. La période de capture est représentée sur la partie inférieure. Dans la zone centrale, sont représentées les différentes activités du système au cours de cette période de capture. Les flèches vertes représentent les interruptions produites, les flèches noires schématisent les changements de contexte entre les tâches. Chaque boîte rectangulaire représente l'exécution d'une tâche pendant une période de temps (la duréee exprimée par la largeur de la boite).

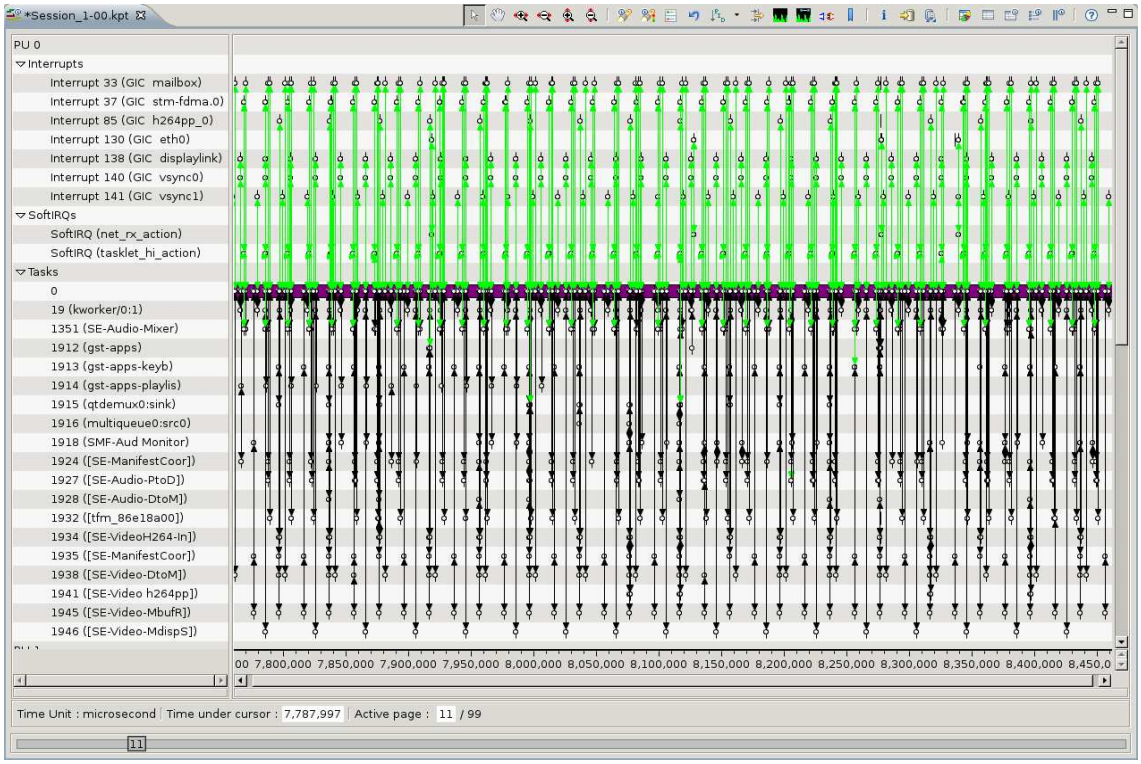


FIGURE 2.9 – Présentation d'une page entière d'événements sur l'outil STLinux Trace Viewer

Face aux problèmes de volumétrie, les techniques de fouille de données se présentent comme une solution. En effet, elles permettent d'extraire des informations pertinentes dans de grandes quantités de données. Afin d'identifier automatiquement les problèmes liés à l'exécution des applications, quelques approches de fouille de données ont été proposées. C'est le cas de l'approche proposée par Lagraa et al. [LTP14] qui permet de détecter automatiquement, à partir des traces d'exécution, les points chauds (hotspots) des applications parallèles. Cependant cette approche ne fonctionne que sur un type précis de trace d'exécution, les traces de bas niveau, autrement dit les traces constituées des événements générés par les composants matériels de l'architecture d'exécution. De plus, le problème adressé dans cette approche est un problème assez spécifique le contexte de l'analyse d'application, celui de l'identification des problèmes de passage à l'échelle des applications parallèles sur les architectures embarqués.

Une autre approche de fouille de données proposée pour l'analyse des applications, en particulier pour les applications multimédia, a été proposée par Lopez-Cueva et al. [CBT+12]. Cette approche permet de détecter automatiquement des ruptures de périodicité de certains composants. Ces ruptures permettent d'identifier les problèmes de décodage de flux audio/vidéo. Cette approche tout comme la précédente ne permet d'identifier qu'un type de problème précis, de plus, elle ne passe pas à l'échelle sur des traces d'exécution de volume important.

Le constat qui se dégage à l'issue de cette analyse est que les outils actuels utilisés pour l'étude des traces d'exécution dans un contexte d'analyse d'applications adressent des problèmes bien spécifiques liés aux applications mais présentent des difficultés d'analyse sur les traces d'application de taille importante. Ceci constitue une difficulté pour l'analyse des traces d'exécution produites par les tests d'endurance. Ces tests visent à assurer la qualité de fonctionnement des applications embarquées destinées à un usage en

continu. C'est généralement le cas des applications multimédia qui équipent les panneaux électroniques d'affichage, les présentoirs vidéo ou encore les set-top box qui affichent en continu des flux vidéo. Les tests d'endurance génèrent des centaines de giga-octets de données. À l'issue de ces tests, il est intéressant d'utiliser plusieurs méthodes afin d'effectuer une analyse plus complète des applications. Malheureusement les techniques actuelles ne passent pas l'échelle sur des traces de grandes quantités et de plus sont très spécifiques.

Face au problème de volumétrie des données contenus dans les traces d'exécution, il serait intéressant de réduire les données des traces pour pouvoir les analyser de manière efficace avec les outils existants. Une réduction effectuée de manière intelligente faciliterait l'utilisation des outils d'analyse de trace d'exécution et réduirait considérablement le temps d'analyse et de mise sur le marché des applications sur les architectures embarquées. Ainsi, nous pensons qu'une méthode de réduction adéquate doit extraire d'une trace d'exécution des données pertinentes pour des analyses minutieuses ultérieures.

2.5 Conclusion

Dans ce chapitre, nous avons présenté l'évolution des architectures des systèmes embarqués, depuis les micro-contrôleurs jusqu'aux MPSoCs qui sont des architectures parallèles et hétérogènes. Cette évolution a été principalement influencée par les exigences des applications en terme de puissance de calcul. Toutefois, cette évolution a complexifié les architectures des systèmes embarqués et leur programmabilité.

L'évolution des systèmes embarqués s'est faite en co-design avec les applications, notamment les applications multimédia. Les systèmes embarqués orientés applications multimédia à l'exemple des baladeurs MP3, des set-top box ou des écrans publicitaires ont ainsi vu le jour. Cependant, ces applications font intervenir des tâches complexes (décodage vidéo, décodage audio, encodage, ...) dont le portage sur les systèmes embarquées est une tâche complexe.

Le développement des applications sur les MPSoCs implique des phases importantes d'analyse et de validation. À cet effet, les méthodes permettant l'analyse sur ces environnements repose sur les traces d'exécution. Toutefois, ces méthodes d'analyse ne passent pas du tout à l'échelle sur des traces de taille importante.

Il est nécessaire d'effectuer au préalable une réduction du volume de données de la trace d'exécution pour pouvoir utiliser correctement les méthodes d'analyse existantes.

CHAPITRE 3

Etat de l'art

Sommaire

Dans ce chapitre, nous présentons dans la section 3.1 les travaux antérieurs effectués dans le domaine de la réduction de données et principalement dans le domaine de l'analyse des applications. Par la suite, nous présentons dans la section 3.2 les travaux proposés pour la détection d'anomalies orientée analyse d'applications.

3.1 La réduction des données

Les méthodes d'analyse de traces d'exécution basées sur des techniques de fouille de données sont généralement très coûteuses en puissance de calcul et à cet effet, elles ne passent pas à l'échelle sur de gros volumes de données. Une solution consiste à réduire les données de manière à ne sélectionner que celles pertinentes pour ces méthodes d'analyse.

La réduction des données constitue généralement une étape de pré-processing dans le processus d'analyse des données. Pour cela, elle doit garantir la pertinence entre l'information contenue dans les données réduites et celle contenue dans les données d'origine [HKP11].

3.1.1 Les méthodes générales de réduction

De nombreuses méthodes ont été proposées pour la réduction des données. Elles se regroupent en trois grandes stratégies : *la compression de données*, *la réduction des occurrences* et *la réduction de la dimensionalité*.

La compression de données est une stratégie qui repose sur des transformations visant à obtenir un *représentation* réduite (compressée) des données d'origine. Parmi les méthodes basées sur cette stratégie, on distingue *LZ78* [ZL78] ou encore *le codage de Huffman* [Huf52]. Ce sont des méthodes générales utilisables

sur n'importe quel type de données. Elles sont utilisées dans les outils de compression comme *7-zip* ou *gzip*. Elles produisent de bons résultats de réduction de données, mais la représentation réduite des données n'est pas utilisable ou compréhensible par les outils d'analyse de données.

La réduction des occurrences est une stratégie qui consiste à remplacer les données d'origine par une forme condensée. Suivant cette stratégie de réduction, on retrouve les approches paramétriques telles que les méthodes de régression linéaire ou les méthodes log-linéaires, et les approches non paramétriques telles que les histogrammes, l'échantillonnage, le clustering ou l'agrégation. Les approches paramétriques construisent un modèle d'estimation des données pour substituer aux données les paramètres du modèle. C'est le cas des méthodes de régression linéaire [NKNW96] dont le modèle s'exprime sous forme d'une variable aléatoire y, appelée *variable réponse*, fonction linéaire d'une variable x (*variable prédictive*),

$$y = wx + b$$

où x, y sont les attributs des données et w, b les coefficients de réduction. Dans le cas des méthodes log-linéaires, les données sont exprimées sous forme de distributions de probabilité puis approximées par des produits de distributions suivant des sous-dimensions données [FR+12]. Les approches non paramétriques consistent à extraire des données d'origine un modèle ou une représentation qui résume ces données. C'est le cas des histogrammes qui partitionnent les données en sous-ensembles disjoints (*buckets*) [Poo97], du *clustering* qui permet de regrouper les données en clusters en fonction de leur similarité, ou encore de l'échantillonnage qui repose sur la sélection d'un sous ensemble de données présentant (au mieux) toutes les propriétés de l'ensemble des données.

La réduction de la dimensionalité regroupe les méthodes dont le principe consiste à réduire le nombre d'attributs considérés dans les données. Parmi ces méthodes, on distingue les méthodes basées sur les transformées en ondelettes dont l'idée consiste à transformer les vecteurs de données en vecteurs de coefficients d'ondelettes à partir desquels un sous-ensemble de vecteurs pertinents est extrait [PTVF07] ; les méthodes basées sur l'analyse des composantes principales dont le principe repose sur la projection d'un jeu de données à n dimensions dans un espace de k vecteurs orthogonaux à n dimensions ($k \leq n$), lesquels fournissent une approximation du jeu de données [NKNW96]. On peut aussi citer comme méthode de réduction de dimensionalité la sélection de sous-ensembles de *features* (caractéristiques) dont l'idée repose sur l'identification et la suppression des attributs non pertinents ou redondants dans le jeu de données [DL97].

Ces approches générales de réduction de données ne sont pas appropriées pour la réduction à la volée des traces d'exécution du fait qu'elles ne passent pas à l'échelle sur de grandes quantités de données (plusieurs centaines de gigaoctets). De plus, la complexité associée à ces approches notamment le cas de la sélection des features ne permet pas leur application à la volée. Enfin, les données produites par ces approches sont difficiles à exploiter et à interpréter. Dans la section suivante nous nous intéressons aux approches de réduction des données proposées dans le contexte de l'analyse des applications.

3.1.2 La réduction des données et l'analyse d'application

Face aux difficultés qu'introduit le volume des données générées pour l'analyse des applications, de nombreuses approches de réduction ont été proposées. Ces approches s'intéressent globalement aux logs d'applications ou encore aux traces d'exécution d'applications.

3.1. La réduction des données

Certaines approches de réduction de traces d'exécution telles que [MM03] et [SS07] adaptent des solutions traditionnelles de compression de données telles que *gzip* [ZL78] pour compresser les adresses d'instructions et de données présentes dans les traces d'exécution (respectivement les lignes de logs d'événements) non pas à des fins d'analyse, mais pour des besoins de stockage. Nous ne parlerons pas de ces méthodes car elles ne présentent aucun intérêt pour l'analyse des applications étant donné que leur exploitation requiert une phase de décompression, ce qui ramène les traces à leur taille d'origine.

Pour la suite, nous regrouperons les approches de réduction suivant deux catégories : les approches de réduction proposées dans un contexte d'analyse de logs réseaux ou systèmes, et les approches de réduction proposées dans un contexte d'analyse de traces d'exécution.

Approches de réduction des logs réseaux et systèmes

Dans le domaine de l'analyse des logs réseaux et systèmes, Hätönen et al. [HBK+03] présentent l'une des premières méthodes de compression pour les logs de réseaux de télécommunications. Cette méthode repose sur l'extraction des motifs fréquents pour capturer les informations dont l'importance est caractérisée par la redondance dans les logs. Ainsi pour chacun des motifs extraits (itemsets fermés fréquents), les auteurs évaluent la couverture sur l'ensemble des données. Suivant cette évaluation, un itemset fermé fréquent est *informatif* s'il possède une bonne couverture. Les motifs informatifs sont utilisés par la suite pour ré-écrire les logs réseaux. Cette méthode présente quelques limitations qui proviennent principalement de la qualité de réduction et de l'approche d'évaluation de l'information. En effet, pour des seuils des paramètres (fréquence ou couverture) élevés, la méthode ne réalise aucune compression. De plus, l'approche d'évaluation de l'information ne porte aucun intérêt aux motifs non-fréquents. Ces derniers peuvent être porteurs d'information dans un contexte d'analyse, c'est le cas des motifs associés aux événements d'erreur ou d'alerte qui ne sont pas fréquents dans les données mais dont la présence indique des situations de dysfonctionnement, d'erreur ou d'exception.

Lu et al. dans [LXY03] présentent une approche de réduction de logs d'applications web dans laquelle l'intérêt est porté sur les profils d'utilisateurs. Les données de profils utilisateurs dans les logs d'applications présentent généralement une forte dimensionalité liée aux différentes pages web accédées. Une URL de page web constitue un attribut et donc une dimension du profil d'accès. Cette forte dimensionalité constitue une difficulté pour l'analyse et l'interprétation des données de profils. Les auteurs proposent une approche de réduction de la dimensionalité dont l'idée repose sur la factorisation des matrices non négatives. Une matrice non négative $X = (x_{ij})_{m \times n}$ est constituée de n vecteurs binaires x_j, chaque vecteur $x_j = (x_{1j}, x_{2j}, \ldots, x_{mj})^T$ représente une session utilisateur où chaque attribut de la session constitue une dimension du vecteur. $x_{ij} = 1$ si l'utilisateur a accédé à la i^{eme} ressource durant la j^{eme} session, sinon $x_{ij} = 0$. Cette factorisation permet de construire des matrices de dimensions réduites. De ces dernières sont extraits les vecteurs de dimensions réduites suivant lesquels les vecteurs de session x_j sont projetés. Par la suite les projections des vecteurs de session sont regroupées suivant d'un algorithme de clustering basé sur *k-means* [HKP11] pour résumer les sessions utilisateurs suivant leur similarité. Cette approche présente un coût de calcul important associé à la factorisation des matrices non négatives, et ne passe pas à l'échelle sur des traces d'exécution de grande taille. De plus, cette approche présente les limitations liées aux approches de clustering notamment le fait d'associer toute donnée à un cluster même si une donnée ne présente aucune similarité avec celles des clusters.

Toujours sur la base de la similarité pour la réduction des données, Christensen et al. [CL13] proposent un schéma de compression adaptatif sur les données de logs d'applications. Le schéma de compression adaptatif considère que l'ensemble des entrées du log D à analyser se structure en plusieurs groupes disjoints $B_1, ..., B_g$ tel que pour toute entrée $e_i \in D$, il existe $j \in [1, g]$, tel que $e_i \in B_j$. Les entrées sont enregistrées dans chacun des groupes suivant leur ordre de production. Une fenêtre glissante est utilisée pour maintenir les entrées les plus récentes de chacun des groupes. Pour déterminer l'appartenance d'une entrée e_i du log à l'un des groupes B_j, un score de similarité est calculé entre e_i et les entrées de la fenêtre glissante de chacun des groupes B_j. Le groupe B_j pour lequel e_i présente le meilleur score de similarité est choisi pour accueillir e_i. Les entrées les plus anciennes des différents groupes peuvent être compressées au fur et à mesure et séparément suivant différentes techniques de compression. Cette méthode présente l'avantage d'organiser les entrées des logs en fonction de leur similarité avant d'y effectuer une compression. Cependant elle requiert la connaissance de tous les groupes d'entrées auxquels toutes les entrées appartiennent, ce qui n'est pas toujours le cas sur les logs d'événements d'applications, notamment celles qui contiennent des anomalies.

Approches de réduction des traces d'exécution

Dans le cas de l'analyse de traces d'exécution, Hamou-Lhadj et al. [HLL02] proposent une approche dans laquelle l'intérêt est porté sur les traces applicatives, c'est à dire des traces constituées d'invocation de fonctions et de méthodes d'objets. Ils construisent une hypothèse suivant laquelle les traces d'exécution sont principalement constituées des invocations répétitives des méthodes et des fonctions. Pour cela, ils construisent une méthode de compression sans perte qui explore les séquences d'appels de méthodes et fonctions pour construire un arbre d'appels. L'arbre d'appels est transformé par la suite en un graphe acyclique direct, où toutes les séquences similaires d'appels sont représentées une seule fois. Le graphe acyclique direct est construit à partir de la technique d'extraction de la sous-expression commune [FSS90]. Cette méthode présente des résultats intéressants, notamment un taux de compression de 31%. Toutefois, elle repose uniquement sur des informations de trace de haut niveau qui ne constituent pas la majorité des données contenues dans les traces d'exécution, notamment dans notre cas, les traces issues de systèmes embarqués. Ces dernières contiennent une grande quantité d'événements systèmes pour lesquels leur approche n'est pas adaptée.

Toujours sur les traces applicatives, Zaidman et al. [ZD04] proposent une heuristique pour structurer une trace d'exécution en clusters afin de regrouper les événements qui partagent la même information. Leur approche repose sur l'analyse de la collaboration entre les composants de l'application. Pour modéliser cette collaboration, les auteurs analysent les échanges de messages suivant l'hypothèse que les messages qui contribuent à la réalisation d'une fonctionnalité présentent plus ou moins la même fréquence. Ainsi, les fréquences d'invocations des méthodes sont calculées et utilisées pour annoter les méthodes d'objets. Par la suite, une opération de clustering est réalisée sur les méthodes annotées pour identifier les groupes de méthodes qui réalisent les mêmes fonctionnalités. L'hypothèse sur laquelle repose la méthode proposée n'est pas toujours correcte car certains messages tels que les messages d'avertissements qui n'ont aucun lien avec une fonctionnalité peuvent avoir la même fréquence que ceux propres à cette fonctionnalité.

Kuhn et al. dans [KG06] proposent une approche de réduction des tailles des traces d'exécution basée sur le regroupement des événements en fonction de leur imbrication. Cette approche assigne à un groupe

les événements consécutifs qui présentent un même niveau d'imbrication. De même, les événements consécutifs qui ont un niveau d'imbrication croissant sont assignés à un même groupe. Un nouveau groupe est initié lorsque le niveau d'imbrication entre deux événements consécutifs décroît au delà d'un certain seuil. Les performances de réduction de cette approche dépendent des niveaux d'imbrications des événements. Il faut noter que cette approche est moins effective sur une trace issue de l'exécution d'une application multi-threadée où il existe une forte interaction entre les threads. De plus, il est difficile de structurer la trace en niveaux d'imbrication.

Amiar et al. dans [ADFdB13] proposent une approche de réduction de traces d'exécution qui repose sur la compression des cycles ou des boucles dans les traces d'exécution. Pour cela, ils font l'observation suivant laquelle les programmes conçus pour les microcontrôleurs sont élaborés autour d'une boucle principale à partir de laquelle ils itèrent au cours de leur exécution. Suivant cette observation, les auteurs proposent une approche de compression qui utilise l'algorithme Sequitur [NMW97] pour identifier les différentes sous-séquences répétitives dans les traces d'exécution. Chaque sous-séquence est réécrite sous la forme d'un symbole non terminal issue d'une grammaire générée à partir de la trace d'exécution.
Exemple : Supposons une trace d'exécution associée à la séquence suivante : $< cabcabcabcabcad >$, l'application de l'algorithme Sequitur sur cette séquence fournit comme résultat la grammaire suivante : $S \rightarrow AABd, A \rightarrow CC, B \rightarrow ca, C \rightarrow Bb$, qui est une réécriture de la trace. Dans cet exemple, nous observons que l'algorithme Sequitur capture explicitement la répétition de la sous-séquence $< cba >$ et fournit une grammaire de 14 symboles pour une séquence de 15 symboles. Pour améliorer le pouvoir de compression de Sequitur, Amiar et al. introduisent la notion de cycle dans les règles de grammaire. Ainsi, lorsqu'une règle modélise une séquence dans laquelle une sous-séquence possède plus d'un cycle, ce nombre de cycles est associé au symbole non terminal associé. D'après l'exemple présentée ci-dessus, la grammaire de compression issue de l'approche proposée est $S \rightarrow cA^4B, A \rightarrow abc, B \rightarrow ad$, qui est plus compacte que celle fournie par Sequitur. Pour être effective, cette approche nécessite la connaissance des événements qui délimitent les boucles dans une trace d'exécution. De plus, cette approche est ineffective sur les traces d'exécution des applications parallèles où il difficile d'extraire des cycles uniformes et pertinents.

Pour mieux adapter les approches de réduction aux techniques d'analyse d'applications, Hamou-Lhadj et al. étudient dans [HLL04] les techniques d'exploitation de données de traces d'exécution par certains outils d'exploration et de visualisation des traces d'exécution. Sur la base de cette étude, ils proposent plusieurs techniques de réduction de traces d'exécution parmi lesquelles le filtrage d'événements dont le principe consiste à filtrer les événements à traiter selon le ou les composants étudiés. Une autre technique proposée est le *pattern-matching* dont le principe consiste à identifier dans les traces des motifs qui correspondent aux motifs de comportements connus afin de les substituer par une forme plus générale. Ces techniques utilisent généralement une connaissance a priori sur les applications et la qualité des résultats dépend fortement des paramètres d'analyse et de la complexité des données de la trace d'exécution.

Dans l'idée d'améliorer l'approche de filtrage pour la réduction des traces d'exécution, Hamou-Lhadj et al. [HLBAL05] proposent une technique de filtrage qui utilise un critère d'utilité pour identifier les composants essentiels dans une trace d'exécution. Ce critère d'utilité repose sur les concepts *Fan-in* et *Fan-out* calculés sur les fonctions. La technique proposée calcule la valeur d'utilité associée à chaque fonction et la compare à un seuil d'utilité. Les fonctions retenues dans la trace d'exécution réduite sont celles qui possèdent une valeur d'utilité supérieure au seuil d'utilité. Cette méthode présente un inconvénient lié à la définition du seuil d'utilité. La formulation du critère d'utilité telle que proposée ne considère pas les

fonctions dont la fréquence d'invocation est faible mais pour lesquelles la présence est pertinente pour l'analyse de l'application, c'est le cas des fonctions associées aux exceptions ou aux erreurs.

Les techniques de réduction de données proposées dans le contexte des traces d'exécution reposent principalement sur l'identification des données redondantes ou des données présentant des caractéristiques communes pour fournir des représentations plus compactes ou plus simplifiées. L'idée générale de ces techniques est construite sur le fait que le volume des traces d'exécution est dominé par des données redondantes. Toutefois, ces techniques ignorent ou ne considèrent pas les événements rares ou exceptionnels dont la présence présente un intérêt essentiel pour l'analyse d'applications notamment pour des fins de débogage, d'optimisation ou de validation. C'est le cas des événements qui décrivent des dysfonctionnements, des irrégularités ou encore des incohérences. En outre, les approches de réduction proposées nécessitent plusieurs passes sur les données et donc ne peuvent être utilisées que post-mortem. Ce qui impose dans ce cas les problématiques liées au stockage des données de traces.

Nous pensons que les événements redondants présentent effectivement un intérêt pour la réduction des traces d'exécution de par le volume considérable qu'ils représentent. Seulement, les événements redondants ne sont pas pertinents dans un contexte d'analyse d'applications dans le sens où ils ne permettent pas de détecter automatiquement des problèmes qui se produisent pendant l'exécution des applications. Ces problèmes d'exécution sont généralement décrits par des événements rares ou des comportements anormaux dont la prédiction est difficile et l'identification compliquée. Les comportements rares et non prédictibles dans le contexte d'analyse d'applications s'avèrent plus pertinents que les comportements redondants. L'identification automatique des cas de comportements rares relève du domaine de la détection d'anomalies.

3.2 La détection d'anomalies

La détection d'anomalies est une discipline du diagnostic dont l'objectif porte sur l'identification d'objets dont les caractéristiques diffèrent de celles attendues. Ces objets aux comportements non conformes sont encore désignés par les termes *anomalies*, *exceptions*, *aberrations* ou *discordances*. De manière générale, une anomalie se définit comme un motif ou une observation qui diffère considérablement de la tendance des données au point de laisser penser qu'il a été généré par un mécanisme différent [HD80].

La détection d'anomalies présente un intérêt important dans de nombreux domaines tels que le contrôle de qualité, la sécurité publique, le traitement des images, la détection d'accidents industriels, la surveillance réseau ou encore l'analyse d'applications.

Dans le domaine de l'analyse d'applications, les anomalies sont des données intéressantes parce qu'elles donnent des informations sur des comportements inattendus ou anormaux, autrement dit, des comportements qui ne respectent pas l'hypothèse de génération des données. Les anomalies se regroupent en trois grandes catégories :

les anomalies globales : ce sont instances de données dont les propriétés dévient significativement par rapport au reste des objets.
Exemple : soient les mesures de durée des différentes instances de la fonction *decode_frame()* au cours de l'exécution de l'application *Player_test* :

instances	time	avg difference
1	$32ms$	$6.142ms$
2	$38ms$	$5.285ms$
3	$35ms$	$4.857ms$
4	$40ms$	$6.428ms$
5	$16ms$	$17.571ms$
6	$39ms$	$5.714ms$
7	$35ms$	$4.857ms$

TABLE 3.1 – Durée de différentes instances de la fonction *decode_frame()*

Suivant la Table 3.1, l'instance d'exécution étiquetée 5 présente une durée qui diffère significativement des durées des autres instances d'exécution, cette instance d'exécution est qualifiée d'anomalie.

les anomalies contextuelles : ce sont les instances de données dont la déviation est jugée significative en fonction d'un contexte spécifique aux objets. Par exemple, la réponse à la question :« Est-ce exceptionnel qu'on ait une température de 28˚C en ce jour ? », dépend des contextes que sont le lieu et la saison. En effet, si le lieu est le Canada et que la saison est l'été, alors ce n'est pas exceptionnel. Par contre si le lieu est le Canada et la saison est l'hiver, alors c'est une anomalie. Le contexte fait partie de la définition du problème. Les anomalies contextuelles sont considérées comme une généralisation des *anomalies locales*. Une donnée est considérée comme une anomalie locale si sa densité dévie significativement des autres données situées dans la zone où elle s'est produite. La qualité de la détection des anomalies contextuelles dans une application dépend fortement de la pertinence des attributs contextuels, et ce en plus de la qualité de la mesure de déviation. Pour cela, les attributs contextuels doivent être définis par les experts du domaine.

les anomalies collectives : elles forment un sous-ensemble du jeu de données pour lequel toutes les données du sous-ensemble dévient du reste du jeu de données entier avec la particularité qu'une seule donnée du sous-ensemble ne peut constituer une anomalie. Par exemple, considérons la séquence d'actions se produisant sur un ordinateur :

... http-web, buffer-overflow, http-web, http-web, smtp-mail, ftp, http-web, ssh, smtp-mail, http-web, ***ssh, buffer-overflow, ftp****, http-web, smtp-mail, http-web, ...*

La séquence d'actions en gras (**ssh, buffer-overflow, ftp**) correspond à une attaque web typique d'une machine distante, suivie d'une copie de donnée de la machine hôte vers une machine distante via ftp [CBK09]. Nous remarquons sur cet exemple que les actions de cette séquence anormale, considérées individuellement, ne constituent pas des anomalies.

De nombreuses approches ont été proposées pour l'identification des anomalies dans les jeux de données. Elles peuvent se regrouper en fonction de la disponibilité des connaissances métier dans la construction du modèle d'identification des anomalies. On distingue :

les approche supervisées qui reposent sur la disponibilité de données d'apprentissage étiquetées par les experts du domaine afin de construire un modèle de détection. Ainsi, toute donnée qui ne correspond pas au modèle est considérée comme une anomalie.

les approches non-supervisées qui supposent l'absence des données fournies par les experts du domaine et se basent sur des hypothèses définies sur les objets pour identifier les anomalies. Par exemple, une

hypothèse est le fait que les objets considérés normaux présentent des caractéristiques communes alors que les objets anormaux présentent des caractéristiques singulières.

les approches semi-supervisées qui reposent sur la disponibilité des données d'apprentissage étiquetées par les experts du domaine qui cependant fournissent des informations partielles, par exemple les classes des données normales, et permettent de construire un modèle partiel de détection. Sur cette base, les données qui ne correspondront pas aux classes construites pendant la phase d'apprentissage seront considérées anormales.

3.2.1 Les méthodes générales de détection d'anomalies

Sous l'hypothèse de la disponibilité des données de références pour l'identification des anomalies ou de la similarité par rapport à l'ensemble du jeu de données, les méthodes de détection d'anomalies peuvent être regroupées en 4 grandes catégories [HKP11] :

1. les approches statistiques,
2. les approches basées sur la classification,
3. les approches basées sur la proximité,
4. les approches basées sur le clustering.

Les approches statistiques (1) font l'hypothèse que les données normales sont générées par un processus stochastique. De ce fait, les données normales apparaissent dans les régions où le processus stochastique présente une forte probabilité et les anomalies dans les régions de faible probabilité. Les approches statistiques se classent en deux grands groupes : les approches paramétriques et les approches non paramétriques. Les approches paramétriques considèrent que les données normales sont générées par une distribution paramétrique estimée (*hypothèse nulle*) dont les paramètres sont calculés à partir des données. Toutes les données qui ne vérifient pas les propriétés de la distribution paramétrique sont considérées anormales [GMES+99, HFLP01]. Pour les approches non-paramétriques, un modèle statistique de « données normales » est élaboré à partir des données d'entrée et toute donnée qui ne correspond pas à ce modèle est considérée anormale [AFV95, DN00]. Les approches statistiques présentent plusieurs inconvénients notamment le fait qu'elles reposent sur des hypothèses statistiques fortes (les approches paramétriques), ou alors qu'il est difficile de construire un modèle statistique précis à partir des données d'apprentissage. De plus l'estimation des paramètres associés aux modèles statistiques peut devenir très complexe surtout dans les cas de modèles sophistiqués (modèles résultant de la combinaison de différents modèles).

Les approches basées sur la classification (2) présentent le problème de détection d'anomalies comme un problème d'attribution de classe (normale ou anormale) aux données sous condition que des données d'apprentissage étiquetées soient disponibles. L'idée générale de ces approches consiste à construire un modèle de classification à partir des données d'apprentissage. Ce modèle de classification est ensuite utilisé sur les données de test pour distinguer les anomalies des instances normales. Les données d'apprentissage contiennent une partie des données étiquetées « normales » et suivant leur disponibilité des données étiquetées « anormales ». Ces approches se structurent en deux groupes, celles basées sur la classification *multi-classes* et celles basées sur la classification *uni-classe*. Les approches de classification *multi-classes* organisent les données normales suivant plusieurs classes et au cours de la phase de détection, les données qui ne correspondent à aucune classe sont des anomalies. Parmi ces approches, nous pouvons citer celle

proposée par Barbara et al. [BWJ01] qui utilise un classifieur bayésien multi-classes pour réaliser la détection d'intrusion dans les réseaux. Les approches *uni-classe* considèrent que les données d'apprentissage constituent une seule classe normale et toute instance de donnée qui ne correspond pas à cette classe est une anomalie. C'est le cas de l'approche proposée par Ratsch et al. [RMSM02] qui utilise les SVM (Support Vector Machine) pour construire un classifieur uni-classe afin de réaliser la détection d'anomalies. La qualité des approches basées sur la classification dépend fortement de la disponibilité et de la qualité des données d'apprentissage étiquetées. Cependant dans certains contextes d'analyse des données, les données étiquetées sont rarement disponibles.

D'autres approches reposent sur des mesures de distance pour quantifier la similarité entre les données, ce sont les *approches basées sur la proximité*(3). Ces approches sont construites sur l'hypothèse selon laquelle la proximité d'un objet anormal par rapport à ses plus proches voisins dévie de la proximité moyenne qui existe entre un objet et l'ensemble des objets du jeu de données. Elles calculent un score d'anomalie à partir d'un rayon r qui définit la portée de voisinage raisonnable pour un objet. Ce voisinage est communément appelé le r-*voisinage*. Un objet est anormal si la taille de son r-voisinage en nombre d'objet est inférieure à un seuil fixé [KNT00]. Une illustration de ces approches de détection d'anomalies est présentée à la Figure 3.1.

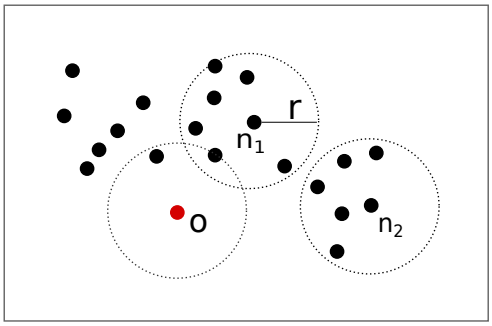


FIGURE 3.1 – Détection des anomalies à partir du r-voisinage : dans cette figure, le seuil du r-voisinage est fixé à 5, les objets n_1 et n_2 sont des objets normaux car r-voisinage(n_1) = 6 et r-voisinage(n_2) = 6, l'objet o est une anomalie car r-voisinage(o) = 2

C'est le cas des techniques de détection d'anomalies proposées par Eskin et al. [EAP+02] et Angiulli et al. [AP02] qui calculent le score d'anomalie pour chacune des instances de données en sommant les distances à leurs k plus proches voisins. Ces approches permettent d'identifier les anomalies globales en ce sens qu'un objet est évalué par rapport à la densité de son voisinage, cette densité étant définie par rapport la distribution globale des données dans le jeu de données. Cependant, certains jeux de données présentent des structures plus complexes où les objets sont considérés anormaux par rapport à leur voisinage plutôt que par rapport à la distribution globale de données. Par exemple, dans la Figure 3.2, il se distingue deux clusters, un cluster dense (C_1) et un cluster peu dense (C_2). L'objet o_3 apparaît comme une anomalie parce qu'il est distant de la majorité du jeu de données. Par contre, il est difficile de considérer o_1 et o_2 comme des anomalies. Si o_1 et o_2 étaient considérés comme des anomalies d'après le principe des approches basées sur la distance, alors tous les objets du cluster C_2 seraient considérés comme des anomalies. Il apparait que o_1 et o_2 sont éloignés du cluster C_2 et plus proches du cluster C_1. Toutefois, ils dévient du cluster C_1. o_1 et o_2 constituent donc des anomalies en fonction de la localité où ils se trouvent, c'est à dire en fonction de C_1. Pour cela, ils sont qualifiés d'anomalies locales.

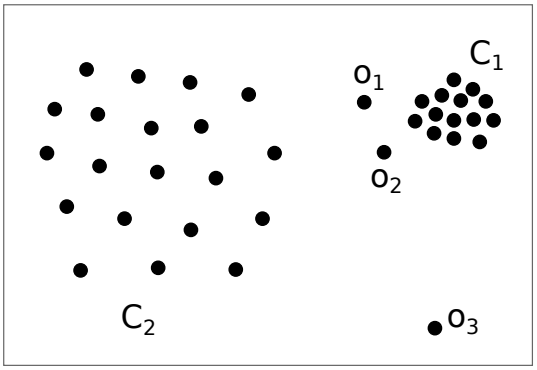


FIGURE 3.2 – Les anomalies globales et locales : o_1 et o_2 sont des anomalies locales, o_3 est une anomalie globale

Breunig et al. [BKNS00] ont proposé l'une des premières méthodes de détection des anomalies locales. Cette méthode compare la densité autour d'un objet par rapport à la densité du voisinage de l'objet à travers un score d'anomalie. Si la densité autour d'un l'objet est similaire à celle de son voisinage, alors l'objet possède un score d'anomalie proche de 1 et est considéré normal, cependant si la densité autour d'un objet est différente de celle de son voisinage alors cet objet possède un score d'anomalie élevé et de ce fait constitue une anomalie. Ces approches présentent quelques contraintes. Dans un contexte d'analyse non supervisée, il est important que les instances normales soient nombreuses et assez proches les unes des autres pour ne pas constituer des fausses anomalies. De plus, les valeurs de score d'anomalies ne sont pas toujours faciles à interpréter. Leur interprétation varie largement d'une application à l'autre.

Les approches basées sur le clustering (4) analysent les relations qui existent entre les données et à partir de ces relations les données sont partitionnées en sous-ensembles denses appelés clusters. Ces clusters sont utilisés pour identifier les anomalies. Pour ces approches, une anomalie se définit :

- soit comme un objet qui n'appartient à aucun cluster, c'est l'hypothèse des approches telles que *DBScan* [EKSX96] ou *SNN* [ESK03],
- soit comme un objet distant de son plus proche cluster, c'est le cas des approches proposées par Labib et al. [LV02], Ramadas et al. [ROT03],
- soit comme un objet qui fait partie d'un cluster peu dense ou de petite taille, c'est le cas de l'approche proposée par He et al. [HXD03].

L'inconvénient important que présentent ces approches est le fait que leur objectif principal repose sur l'identification des clusters et non la détection d'anomalies. La construction de clusters induit un coût important lié aux techniques de clustering. De plus, la qualité de la détection des anomalies de ces approches dépend fortement de la qualité des clusters. Certaines méthodes de clustering utilisées par ces approches forcent tout objet à appartenir à un ou plusieurs clusters, conduisant ainsi à des clusters très larges qui rendent difficile la détection d'anomalies.

Dans la suite de cette section, nous présenterons quelques approches de détection d'anomalies proposées dans le cadre de l'analyse des logs réseaux et dans le contexte de l'analyse d'applications.

3.2.2 Quelques approches de détection d'anomalies pour les logs réseaux

L'analyse des logs réseaux est l'un des domaines d'application pour lequel de nombreuses approches de détection d'anomalies ont été proposées non seulement pour l'identification des problèmes dans les réseaux mais aussi pour la détection d'intrusion.

Les premières approches utilisées pour la détection d'anomalies dans les réseaux sont les approches de détection basées sur les signatures [PP07, KSD09]. Ces approches reposent sur les techniques de comparaison entre la signature ou les motifs associés aux données à analyser et ceux associées aux anomalies connues. Parmi ces approches on peut citer les systèmes de détection d'intrusion *(IDS)*. Bien que ces approches présentent l'avantage de produire un faible taux de fausses alarmes, elles ne détectent pas en général les anomalies dont les signatures sont inconnues. En outre, pour demeurer effectives, les signatures doivent régulièrement être mises à jour.

Face à la difficulté induite par l'utilisation des signatures pour la détection des anomalies, d'autres approches se sont orientées vers la surveillance des changements de comportements et l'étude comportementale des hôtes du réseau à partir de leurs logs. C'est le cas d'Ahmed et al. dans [AOC07] qui proposent une technique d'analyse des changements de distribution du trafic IP à partir des méthodes de clustering. Leur approche est construite sur l'hypothèse que les comportements normaux se regroupent facilement en clusters et qu'une anomalie résulte d'un changement brusque de comportement par rapport à l'ensemble des comportements normaux. Les auteurs élaborent une approche de clustering basée sur une fonction « kernel récursive » qui permet d'identifier les attributs caractéristiques des comportements normaux. Par la suite, toute nouvelle instance de donnée est projectée dans l'espace de caractéristiques des comportements normaux et une erreur de projection est calculée. Lorsqu'une erreur de projection est importante, cela signifie que l'instance de donnée associée provoque un changement important de comportement par rapport aux comportements normaux et constitue de ce fait une anomalie. L'approche proposée repose sur un modèle d'apprentissage dont la détermination des paramètres est complexe et pour cela les auteurs utilisent généralement un kernel gaussien pour estimer la distribution des données. De plus, cette approche de détection est très sensible à la qualité des données d'apprentissage, elle ne tolère pas la présence d'anomalies dans les données d'apprentissage.

Soo-Yeon et al. dans [JCJ14] proposent plutôt une approche d'analyse à deux niveaux des comportements dans les réseaux. Le premier niveau consiste à extraire les caractéristiques propres aux comportements normaux et anormaux à partir d'un jeu d'apprentissage. Les caractéristiques extraites sont utilisées pour construire un modèle de classification et de régression (*CART*). Ce modèle sert de générateur de règles pertinentes pour la détection des anomalies. Le second niveau quant à lui consiste à identifier les anomalies à partir des règles construites et à retrouver les caractéristiques pertinentes auxquelles elles sont associées, ceci dans le but de fournir une interprétation. Cette approche présente une complexité importante et ne passe pas à l'échelle sur des grandes quantités de données.

Ren et al. [RHL+08] proposent aussi une approche basée sur le clustering pour la détection les anomalies dans les réseaux. Leur approche repose sur la caractérisation des clusters associés aux profils normaux de fonctionnement du réseau. Chaque cluster est défini par son cœur (données situées au centre du cluster) et sa frontière (données situées en frontière du cluster). Pour toute nouvelle instance de donnée, l'approche évalue la distance entre cette instance de donnée et les cœurs des clusters, et celle entre l'instance de don-

née et les frontières des clusters. Une instance de données est anormale si elle se trouve à l'extérieur des clusters. Par contre lorsqu'une nouvelle instance de donnée se trouve dans un cluster et influence le cœur de ce dernier, elle déclenche une mise à jour du cœur du cluster et donc du profil réseau associé. L'approche proposée utilise l'algorithme DBScan et présente l'inconvénient associé à ce dernier autrement dit, cette approche ne permet pas de gérer les clusters de densités différentes. De plus, l'évaluation de cette approche n'est réalisée que sur des données relativement simples (faible dimensionalité) auxquelles elle semble bien adaptée. Ce n'est pas le cas des traces d'exécution qui présentent une grande dimensionalité (près de 1000 types d'événements).

Dans le domaine de l'analyse des logs réseaux, Chen et al. dans [CCCH09] s'intéressent principalement aux problèmes d'intrusion. Ils utilisent la théorie des ensembles approximatifs (*rough sets*) et les SVM pour identifier les intrusions. La théorie des ensembles approximatifs est utilisée pour réduire les dimensions des données contenues dans les logs. Par la suite, les caractéristiques sélectionnées sont transmises à un SVM pour l'apprentissage et la détection. Bien qu'effective, la théorie des ensembles approximatifs présente un coût important en fonction de la dimensionalité. De plus, cette théorie fonctionne bien lorsqu'il est facile d'estimer la dépendance entre les dimensions des données, ce qui n'est pas le cas sur les traces d'exécution où il est difficile d'extraire les dépendances entre les attributs des événements et particulièrement dans un contexte d'exécution parallèle. En outre, malgré la puissance de classification des SVM, l'utilisation des SVM passe généralement par le choix d'une fonction kernel prédéfinie appropriée, ce qui est fastidieux car le pouvoir de classification d'une fonction kernel peut fortement varier d'un jeu de données à l'autre.

3.2.3 Les approches de détection d'anomalies sur les traces d'exécution

Dans le domaine de la détection d'anomalies sur les traces d'exécution, les approches proposées se regroupent généralement en deux grandes catégories : les approches (semi-) supervisées et les approches non-supervisées.

Les approches (semi-) supervisées se réalisent en deux phases : une phase d'apprentissage pendant laquelle une trace de référence est utilisée pour construire un modèle caractéristique des comportements normaux de l'application ; et une phase de détection qui consiste à comparer le modèle caractéristique aux traces de test pour détecter les comportements anormaux.

Les approches non-supervisées réalisent la détection des anomalies dans les traces d'exécution d'application sans utiliser une trace de référence. La seconde catégorie présente un intérêt pour l'analyse des trace multimédia vu que les développeurs ne disposent pas toujours des traces de référence pour les applications.

Les approches (semi-) supervisées

L'une des tendances des approches (semi-) supervisées consiste à utiliser des techniques de fouille de données pour construire un modèle de classification à partir de la trace de référence. C'est suivant cette idée que Bose et al. proposent, dans [BS05], une approche qui utilise un SVM uni-classe pour construire un modèle de classification à partir d'une trace de référence. Cette trace de référence contient les exécutions correctes. Pour construire le modèle de classification, les auteurs s'intéressent aux appels systèmes contenues dans les traces d'exécution et les représentent sous forme de *n-grams* pour capturer les motifs locaux

et préserver les dépendances temporelles sur les événements. Les *n-grams* extraits de la trace de référence sont utilisés pour former le modèle SVM uni-classe. Cette approche dépend du type de fonction kernel pour réaliser la classification des données. Les auteurs dans cette approche utilise la fonction kernel *k-spectrum*, bien adaptée à leur modèle de données.

Liu et al. dans [LYY+05] proposent un framework pour les traces d'exécution sous forme de graphes de comportements, afin de capturer la séquence d'appels des fonctions et l'ordre d'invocation des tâches. Le principe de la méthode consiste à représenter plusieurs instances de traces d'exécution d'apprentissage (correctes et problématiques) sous forme de graphes de comportements dans le but d'extraire des motifs résumés (graphes fréquents fermés). Ces graphes fermés sont utilisés pour construire un modèle de classification basé sur un SVM. Cette approche est totalement supervisée car son modèle de classification repose sur les deux classes de données (normales et anormales), de ce fait, cette méthode peut fournir des prédictions erronées dans le cas où les traces de test contiennent des problèmes non répertoriés. De plus le modèle de données utilisé constitue une difficulté pour le passage à l'échelle sur des traces d'exécution de grande taille.

Dans le domaine de l'analyse des applications sur les systèmes embarquées, Zadeh et al. dans [ZSK+14] proposent une technique de détection d'anomalies offline *(SiPTA)* basée sur la périodicité des données dans les systèmes embarqués. La technique proposée analyse le signal produit par les systèmes embarquées à l'aide des méthodes telles que la transformée de Fourier, afin d'extraire les motifs périodiques contenus dans les traces. Les motifs extraits sont transformés en séries temporelles. Cette approche fait usage des données d'apprentissage contenant des motifs périodiques normaux et anormaux dans le but d'identifier les caractéristiques spécifiques aux exécutions problématiques et construire un modèle de classification. Le modèle de classification construit est utilisé pour effectuer la détection sur les données de test. Cette approche présente la difficulté commune aux approches basées sur la classification qu'est la difficulté de détection des anomalies dont les caractéristiques ne sont pas connues par le modèle de détection.

Les approches non-supervisées

Baah et al. [BGH06] utilisent le modèle de Markov observable pour capturer la sémantique partielle d'une application à partir des traces d'exécution. Le modèle de Markov observable est construit à partir des traces d'exécution d'apprentissage. Par la suite le modèle de Markov observable est transformé en un modèle de Markov caché. Ce dernier est utilisé comme modèle de classification pour réaliser la détection des anomalies. La méthode proposée a été testée uniquement sur des traces d'exécution de programme dont la modélisation n'est pas complexe. Cette approche ne permet pas d'identifier les problèmes liés à la propagation des anomalies telles que les erreurs de calculs, à cause de l'hypothèse de Markov de premier ordre qui voudrait que la prévision de l'état futur ne dépende que de l'état présent. À cet effet, une erreur qui s'est produite dans le passé et qui n'est pas perceptible dans le présent ne sera pas détectée dans le futur. Cette approche ne passe pas à l'échelle à cause du nombre d'états qui devient rapidement important suivant la complexité du programme analysé.

Dans le cas de la détection en ligne des défauts logiciels, les méthodes d'analyse sont confrontées à un challenge important dû au flux continu et potentiellement infini de traces d'exécution. De plus, au cours de l'exécution, ces traces d'exécution peuvent couvrir différentes régions du programme et présenter diffé-

rents comportements. Dans ce contexte, certaines techniques telles que l'approche proposée dans [BGH06] effectuent un apprentissage hors ligne et déploient le modèle appris pour la détection d'anomalies en ligne.

Par ailleurs, pour maintenir l'évolution des comportements des applications durant leur exécution, ces techniques utilisent des techniques d'adaptation sur des longues exécutions. C'est le cas de l'approche dans [DGIM02a] qui utilise une fenêtre glissante pour capturer les données récentes du flux de traces d'exécution et maintenir à jour les modèles inhérents. La difficulté associée à la mise à jour des comportements est la fréquence des mises à jour. Une mise jour réalisée à la production de chaque événement permettrait de détecter les changements brusques de comportements correspondant à des anomalies mais induit un coût de traitement non négligeable. Par contre une mise à jour effectuée après une période plus importante permet de détecter uniquement des changements lents masquant ainsi les changements brusques.

Zhang et al. [ZBY+12] proposent une méthode qui capture les caractéristiques des traces d'exécution à travers un arbre préfixe. L'arbre préfixe construit sur une trace d'exécution constitue un résumé de la trace qui peut facilement être mis à jour à la volée et dont la taille croît lentement. Par la suite, les auteurs proposent une définition de la distance entre les nœuds d'un arbre préfixe, puis une méthode de recherche des k plus proches voisins d'un nœud sur une structure d'arbre préfixe. À partir de la nouvelle définition de la distance et de l'algorithme de recherche des k plus proches voisins, les auteurs élaborent et comparent trois techniques de détection d'anomalies (une technique basée sur le clustering, une technique basée sur la densité et une technique basée sur les automates probabilistes). Ils montrent que les nouvelles définitions de la distance et de la recherche des k plus proches voisins réduisent considérablement le temps de calcul sans nuire à la précision.

Wang et al. utilisent dans [WWZ+14] un algorithme de clustering pour réaliser un apprentissage en ligne des comportements des applications. La modélisation du comportement d'une application est liée à sa charge de travail sur les systèmes d'exécution. Cette charge de travail est fonction des modèles d'accès aux ressources des applications et du volume de requêtes. Les comportements sont représentés sous forme de matrices de requêtes et regroupées en fonction de leur similarité. La mesure de similarité utilisé dans cette approche est la distance de *Mahalanalobis* [Mah36]. Pour détecter les charges de travail correspondantes au fonctionnement anormal des applications, les auteurs utilisent l'approche *LOF* (Local Outlier Factor) qui attribue à une charge de travail un score d'anomalies. Cette approche compare la densité autour d'un objet par rapport à celle de son voisinage. Ainsi un objet dont la densité associée est différente de celle de son voisinage est une anomalie. Le score d'anomalie est par la suite évalué suivant un test statistique (test de *Student*) pour quantifier l'importance de l'anomalie. L'utilisation du clustering, notamment la technique *K-means* implique la connaissance au préalable du nombre de clusters contenus dans les données. De plus le clustering tend à rattacher toutes les données à l'un des clusters, même celles qui se présentent comme des anomalies.

Après ce panorama des différentes méthodes de détection d'anomalies pour l'analyse des applications à partir des logs et traces d'exécution, nous présentons dans la suite et de manière détaillée une méthode de l'état de l'art que nous utilisons dans notre approche pour effectuer la détection des anomalies.

3.2.4 La détection des anomalies locales : le Facteur Local d'Anomalie (LOF)

À la section 3.2.1, nous avons vu que les approches qui estiment la déviation d'un objet en fonction de la distribution globale des données permettent uniquement de détecter les anomalies globales et considèrent les anomalies locales comme des instances normales de données. Afin de combler cette insuffisance, des approches ont été élaborées pour mesurer la déviation en fonction des distributions locales contenues dans les données. L'hypothèse de base de ces approches repose sur l'idée que la densité autour d'un objet normal est similaire à la densité autour de chacun de ses voisins, par contre, la densité autour d'un objet anormal est différente de la densité autour de ses voisins. C'est autour de cette hypothèse que Breunig et al. [BKNS00] ont proposé l'une des premières méthodes de détection d'anomalies locales. Cette méthode assigne à chaque objet un score d'anomalie nommé *"Local Outlier Factor"*(LOF). Pour un objet donné, ce score est égal au ratio de la densité moyenne autour de ses k plus proches voisins et de la densité autour de l'objet lui-même. Pour déterminer la densité locale d'un objet o étant donné un ensemble d'objets D, les auteurs calculent le rayon de la plus petite hyper-sphère centrée sur o qui contient ses k plus proches voisins, c'est la $k\_distance(o)$. Une illustration de la notion de $k\_distance(o)$ est présentée à la Figure 3.3.

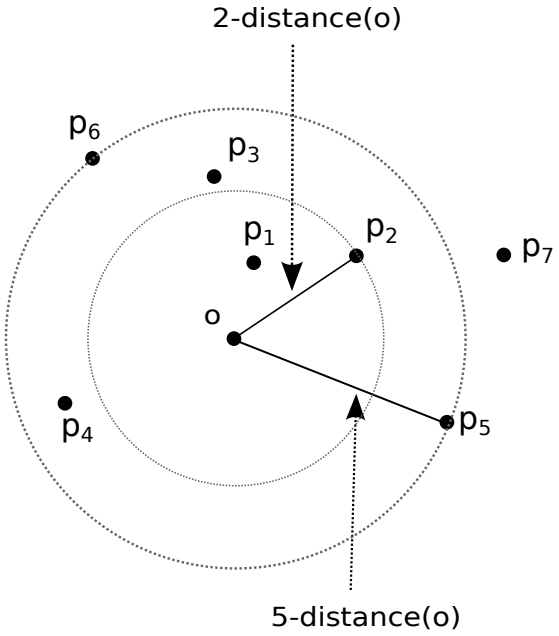


FIGURE 3.3 – Illustration de la notion de k_distance : P_5 et P_6 sont sur la sphère de centre o et de rayon $5\_distance(o)$, P_6 n'est pas dans $5V(o)$ mais dans $N_5(o)$

La $k\_distance(o)$ se présente donc comme la distance entre un objet o et son k^{eme} plus proche voisin.

Définition 1. *Soit $KV(o)$ l'ensemble des k plus proches voisins de l'objet o, lxa k_distance d'un objet o, k_distance(o), se définit formellement comme :*

- *les voisins du point o sont dans l'ensemble de tous les objets*
- $k\_distance(o) = distance(o, p)$ *avec*
  - $p \in KV(o)$
  - $p = argmax_{x \in KV(o)} dist(o, x)$

Cette distance permet de définir la notion de k-voisinage d'un objet o.

Définition 2. *Le $k\_voisinage(o)$ d'un objet o, noté $N_k(o)$, est l'espace qui contient uniquement les points situés à une distance inférieure à la $k\_distance(o)$ par rapport au point o :*

$$N_k(o) = \{o' | o' \in D, distance(o, o') \leq k\_distance(o)\}.$$

Le $k\_voisinage(o)$ d'un objet o peut contenir plus de k objets car plusieurs objets peuvent être sur l'enveloppe de la sphère centrée sur o.

Le $k\_voisinage(o)$ constitue l'ensemble des objets à partir duquel la densité locale associée à l'objet o est déduite. Elle peut être mesurée en calculant la moyenne des distances entre les objets du voisinage et l'objet o. Cependant les différentes valeurs de distances associées au $k\_voisinage$ peuvent induire des fluctuations statistiques importantes dans l'expression de la densité locale. Pour cela, les auteurs ont introduit un mécanisme de lissage : la *distance d'accessibilité* ou *reachability distance*.

Définition 3. *La distance d'accessibilité entre deux objets o et o' est la $distance(o \leftarrow o')$ telle que :*

$$reach\_distance_k(o \leftarrow o') = max\{k\_distance(o), distance(o, o')\}.$$

La distance d'accessibilité est une distance non-symétrique :
$reach\_distance_k(o \leftarrow o') \neq reach\_distance_k(o' \leftarrow o)$.

La distance d'accessibilité permet de définir une densité locale plus robuste à la variation des valeurs de distances du voisinage. Ainsi, les objets qui partagent le même rayon de voisinage doivent présenter une densité locale égale.

Définition 4. *La densité locale ou encore densité d'accessibilité locale d'un objet o, notée $lrd_k(o)$, est :*

$$lrd_k(o) = \frac{\| N_k(o) \|}{\sum_{o' \in N_k(o)} reach\_distance_k(o' \leftarrow o)}.$$

Le score d'anomalie LOF est le rapport entre la densité locale d'un point et la moyenne des densités locales de son voisinage.

Définition 5. *Le score d'anomalie LOF d'un objet o, noté $LOF_k(o)$, est :*

$$LOF_k(o) = \frac{\sum_{o' \in N_k(o)} \frac{lrd_k(o')}{lrd_k(o)}}{\| N_k(o) \|}.$$

Le score d'anomalie LOF d'un objet o se présente comme le ratio entre la moyenne des densités locales des k-plus proches voisins de o et la densité locale autour de l'objet o. Sur la Figure 3.4, pour une valeur de k fixée à 3, o_2, o_3 et o_4 sont les 3-plus proches voisins de o_1. Nous remarquons que la $3\_distance(o_1)$ (représentée par d_{o_1} sur la Figure 3.4) est supérieure à la moyenne des $3\_distance(o_2)$, $3\_distance(o_3)$, $3\_distance(o_4)$, distances représentées respectivement par d_{o_1}, d_{o_2}, et d_{o_3} sur la Figure 3.4. Ainsi la densité locale $lrd_k(o_1)$ est plus faible que celles de ses plus proches voisins. Par conséquent le score d'anomalie LOF associé à o_1 est élevé (> 1). En conclusion, o_1 n'appartient pas à la zone dense où se trouvent ses plus proches voisins ; o_1 est une anomalie.

Une instance de donnée qui se situe dans une région dense, autrement dit, qui a une densité similaire à celle de ses plus proches voisins, présente une valeur de LOF approximativement égale à 1. De ce fait,

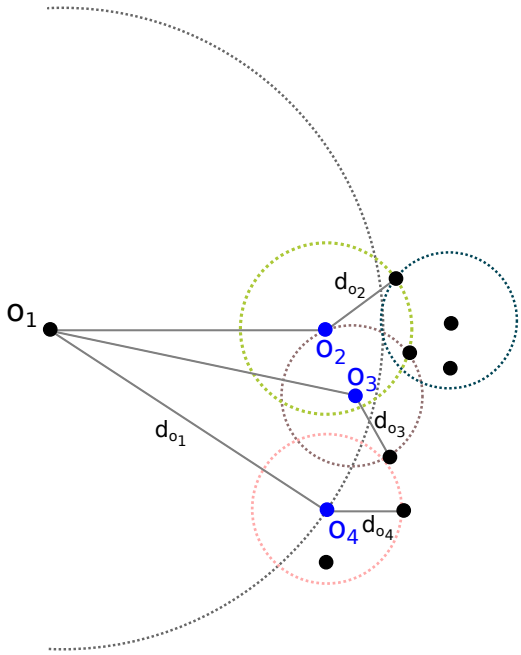


FIGURE 3.4 – Illustration de l'idée de base de la valeur LOF

elle est considérée comme une instance normale. Par contre, une instance de donnée qui se situe dans un cluster peu dense ou à l'extérieur d'un cluster dense est considérée comme une anomalie. Sa densité locale inférieure à celle de ses plus proches voisins et donc sa valeur de LOF est élevée.

Une illustration d'un ensemble d'objets dans un espace à deux dimensions est présentée en Figure 3.5. Il apparaît de manière grossière deux clusters dans les données (C_1 et C_2). Le calcul des scores d'anomalie (valeur de LOF) sur les données montre effectivement que les objets contenus dans les clusters de données présentent des valeurs de LOF proches de 1. Par contre, plus un objet est éloigné de l'un des clusters, plus son score d'anomalie est élevé.

La détection d'anomalie par la mesure de densité locale notamment à travers le calcul des LOFs présente quelques inconvénients, notamment le fait qu'il est difficile de déterminer à l'avance le seuil de LOF permettant de discriminer les anomalies des instances normales, ou encore le fait qu'elle n'associe aucune interprétabilité sur les anomalies. Pour apporter des solutions à ces manquements, des travaux ont été proposés pour améliorer la détection des anomalies à travers LOF sur plusieurs aspects.

Sur le principe associé au calcul du scores d'anomalie LOF, Papadimitriou et al. proposent dans [PKGF03] une mesure appelée *Facteur de Déviation Multi-granularité* (MDEF) qui est une variation de LOF et fournit une importance statistique de la déviation associée à un objet. Cette approche consiste à évaluer l'écart-type des densités locales des plus proches voisins d'un objet (y compris celle de l'objet). L'inverse de la déviation standard constitue alors le score d'anomalie pour une instance de donnée. Outre la détection d'anomalie, cette approche permet aussi de découvrir les micro-clusters d'anomalies. L'algorithme proposé, appelé *LOCI*, nécessite la définition d'un rayon de voisinage dans le but de réduire la sensibilité face au choix de la taille du voisinage définie par l'approche LOF. Pour cela, ce rayon de voisinage est utilisé pour déterminer un échantillon de voisins plus proche à partir duquel la valeur MDEF sera calculée. Toutefois, le choix

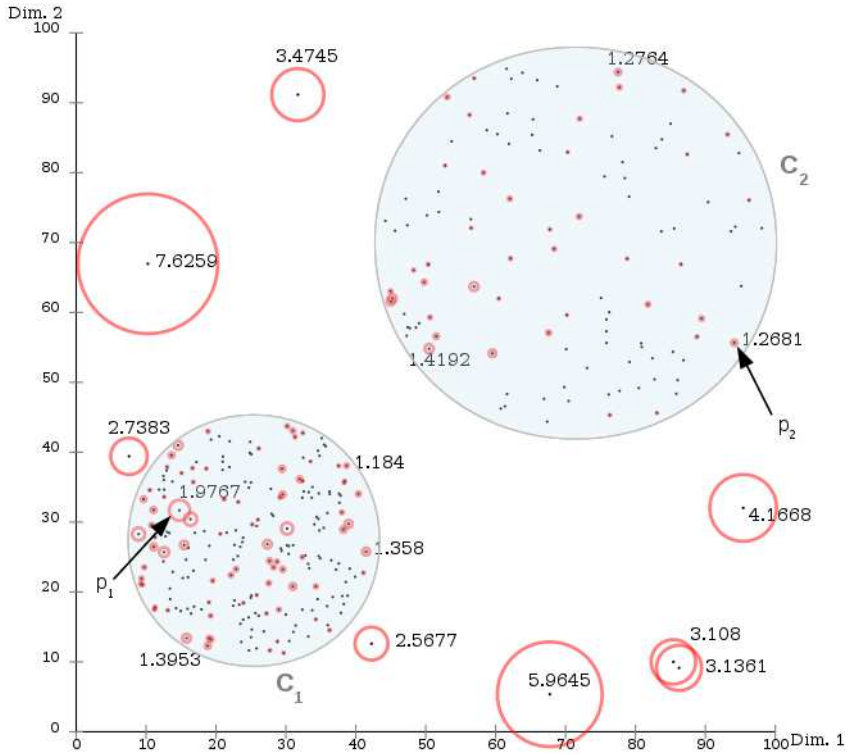


FIGURE 3.5 – Illustration des scores de LOF dans un jeu de données : les cercles de couleurs rouges représentent les valeurs de LOF associées aux objets. Dans le cluster C_1, l'objet p_1 présente une valeur de LOF élevée (1.9767) car bien qu'il se trouve dans un cluster dense, la densité autour de ce point est plus faible par rapport à celles autour de ses plus proches voisins. D'un autre côté, l'objet p_2 qui se trouve dans le cluster peu dense C_2 possède une densité similaire à celles autour de ses plus proches voisins et donc une valeur de LOF (1.2681) proche de 1.

d'une valeur appropriée pour le rayon conditionne les performances de détection de l'algorithme surtout pour les jeux de données à forte dimensionalité. Par ailleurs, Latecki et al. dans [LLP07] montrent que *LOCI* a tendance à considérer les données en frontière de clusters comme des anomalies, contrairement à l'approche LOF.

Tang et al. dans [TCFC02] s'intéressent à la connectivité entre les voisins d'un point en plus de leur densité pour définir le score d'anomalies. L'idée de leur approche consiste à exprimer la déviation (isolativité) comme le degré de déconnexion qu'un objet présente par rapport à ses plus proches voisins. Une haute isolativité implique une faible densité, mais une faible densité n'implique pas forcement une haute isolativité. Le score d'anomalie selon cette approche est exprimé en fonction du ratio entre la moyenne des distances d'un point à ses plus proches voisins et la moyenne des distances pour chacun des k plus proches voisins du point à ses propres k plus proches voisins. Ce score d'anomalie est nommé *COF*(Connectivity-Based Outlier Factor). Cette approche permet de détecter les anomalies qui dévient d'une certaine régularité dans le voisinage (Figure 3.6) et pas uniquement en fonction de la densité.

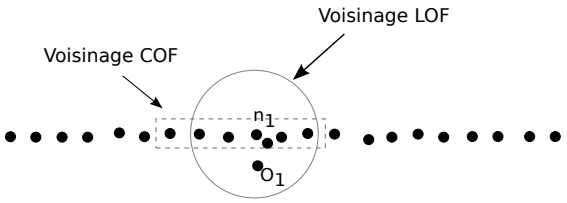


FIGURE 3.6 – Différence entre les voisinages calculés par COF et LOF

Cette approche permet une détection plus effective que l'approche LOF mais par contre elle est plus coûteuse en calcul avec une complexité en $O(N^2)$ sur les jeux de données à grande dimensionalité, N représentant le nombre de dimensions.

Hautamäki et al. dans [HKF04] proposent une version plus simple de LOF qui calcule une valeur appelée *ODIN (Outlier Detection using In-degree Number)* pour chaque objet. L'ODIN d'un objet o est égal au nombre des k plus proches voisins de o qui ont chacun o dans la liste de leurs k plus proches voisins. Cette approche est adaptée pour une modélisation des données sous forme de graphes.

Certaines approches s'intéressent à l'une des difficultés inhérentes aux techniques de détection d'anomalies basées sur le calcul des distances sur l'ensemble des dimensions de l'espace de données. En effet, dans des espaces à très grande dimensionalité, les données peuvent être très dispersées et le concept de similarité ou distance peut devenir moins significatif [BGRS99, AY01]. À cet effet, Lazarevic et al. se proposent dans [LK05] de calculer le valeurs LOF sur des sous-ensembles de dimensions puis par la suite de combiner les résultats, ceci dans le but d'améliorer la qualité de détection dans les espaces à très grandes dimensions.

Pour améliorer le problème d'interprétabilité lié aux valeurs de LOF, Schubert et al. proposent dans [SWZK12], un framework qui combine les prédictions de plusieurs algorithmes de détection d'anomalies à travers l'apprentissage par ensemble dans le but de classifier les scores d'anomalies en fonction de leur pertinence sur les données.

Des approches telles que [SC04, CS06, YQLZ06] proposent différentes variantes de LOF permettant de

manipuler de manière efficiente différents types de données telles que les données spatiales. Pour cela ces approches utilisent des mesures de similarités à l'endroit des distances dans le but de considérer les attributs catégoriques des données.

Pokrajac et al. s'intéressent dans [PLL07] à une version incrémentale de la méthode LOF pour l'adapter à l'analyse des flux tels que le trafic réseau, les trafics de contrôle et de surveillance. L'approche proposée consiste à mettre à jour automatiquement les profils des données par des opérations d'insertion et de suppression des données tout en maintenant à jour le calcul des densités associées aux données. Cette approche ne correspond pas à notre modèle de détection car les profils des données de référence ne changent pas et de ce fait n'ont pas besoin d'être mis à jour.

Conclusion

Les approches de réduction de traces d'exécution s'intéressent généralement aux données redondantes dans les traces. Ces approches ne considèrent pas les données rares ou non redondantes dans leur modèle de réduction. Toutefois dans certains contextes telles que l'analyse d'applications, les événements rares ou non redondants apparaissent plus pertinents que les événements redondants. En effet, les événements redondants représentent principalement les propriétés ou comportements connus d'une application alors que les événements rares ou irréguliers représentent les propriétés ou comportements inconnus, voire inattendus dans le fonctionnement d'une application. Ils correspondent généralement à des anomalies de fonctionnement.

La détection d'anomalies se présente comme une approche intéressante pour l'identification des données rares ou non redondantes dans de grands jeux de données. Dans le domaine de l'analyse des logs ou traces d'exécution, de nombreuses approches de détection d'anomalies ont été adaptées pour identifier les comportements anormaux des applications. Les techniques de détection qui présentent le plus d'intérêt dans le contexte d'analyse d'applications sont les techniques qui ne requièrent pas des données d'apprentissage et analysent les propriétés des données pour extraire celles qui dévient de manière importante de la majorité des données. Parmi ces techniques, l'approche LOF nous paraît la plus intéressante. Elle compare une donnée non pas à l'ensemble du jeu de données mais aux instances données similaires pour déterminer si cette donnée est une anomalie ou non. Cette approche a été à la base de nombreuses techniques plus élaborées telles que [PKGF03, TCFC02, LK05, PLL07], cependant ces techniques n'améliorent que de très peu le capacité de détection de l'algorithme de base LOF mais ajoutent un surcoût de calculs non négligeable au processus de détection. Nous souhaitons réduire les traces d'exécution à la volée à travers la détection d'anomalie qui permettrait de ne retenir que les parties de la traces qui présentent un intérêt pour l'analyse d'application et la méthode LOF nous semble appropriée pour réaliser la détection d'anomalie à la volée.

CHAPITRE 4

Réduction en ligne de traces d'exécution

Sommaire

Dans ce chapitre, nous nous intéressons au problème de réduction en ligne du volume des données des traces d'exécution. Nous proposons une approche de réduction à la volée des traces d'exécution basée sur la détection des anomalies. Cette approche repose sur l'analyse des traces d'exécution au fur et à mesure qu'elles sont générées dans l'objectif d'identifier et de retenir les parties qui présentent un intérêt pour le débogage ou le profilage. Pour ce faire, notre approche s'établit en deux étapes : la première consiste à apprendre automatiquement les comportements *réguliers* d'une application ou du système d'exécution pour lequel la trace est générée ; la seconde repose sur la détection à la volée des comportements *irréguliers* de l'application tracée. Ces comportements *irréguliers* se définissent comme ceux qui dévient des comportements réguliers et qui sont susceptibles de décrire ou de révéler des problèmes d'exécution.

4.1 Contexte

Les traces d'exécution constituent le principal outil de débogage et de profilage sur les architectures d'exécution complexes telles que les systèmes embarqués. Néanmoins elles introduisent une difficulté significative : leur volumétrie qui peut devenir très vite élevée. Ainsi, suivant certains cas d'utilisation et selon le niveau de traçage d'une application sur les systèmes embarqués, les traces d'exécution peuvent facilement atteindre plusieurs centaines de giga-octets voire des téraoctets. Par conséquent elles peuvent rapidement devenir difficiles à analyser avec les méthodes et outils existants dans le domaine de l'analyse des applications.

Face au problème de la volumétrie des traces d'exécution, plusieurs approches de réduction des données ont été proposées. Cependant, ces approches s'adaptent mal aux activités de débogage, de profilage ou de validation des applications. En effet, certaines approches produisent comme résultat des traces compressées sous un format complexe à exploiter dans le cadre de l'analyse des applications. D'autres approches se focalisent sur l'identification et la réécriture des motifs redondants pour exprimer les régimes réguliers des applications. Elles ont pour but de fournir une représentation compressée des traces d'exécution à partir des motifs redondants. Toutefois, ces approches ne prennent pas en compte les événements non redondants ou rares qui peuvent être présents dans les traces d'exécution. Ces événements sont pour la plupart pertinents dans un contexte d'analyse des applications en ce sens qu'ils contiennent des informations décrivant les pro-

blèmes de fonctionnement de ces dernières. Il est donc intéressant d'identifier et considérer ces événements rares dans une approche de réduction des traces d'exécution à des fins d'analyse d'applications.

Motivées par l'identification des informations inattendues ou non redondantes dans les traces, des approches d'analyses de traces d'exécution ont été élaborées à partir des techniques de détection d'anomalies. Parmi ces approches, certaines reposent sur la disponibilité de données d'apprentissage étiquetées par les experts pour identifier dans les traces d'exécution les périodes de fonctionnement d'une application qui diffèrent de ceux attendus. D'autres élaborent des modèles permettant de capturer les propriétés redondantes associées aux fonctionnements normaux des applications. Ces propriétés sont utilisées pour identifier les régimes de fonctionnement exprimés par les données non redondantes et qui représentent des périodes de fonctionnement inattendu. Bien qu'appropriées pour l'analyse des applications, ces approches présentent une complexité importante en calcul et ne peuvent être utilisées que de manière post-mortem sur les traces d'exécution, ce qui ne résout pas les difficultés liées au stockage des énormes volumes de traces d'exécution.

L'approche que nous proposons repose sur l'identification à la volée des régimes de fonctionnement d'une application à partir de sa trace d'exécution. Notre objectif consiste à identifier et stocker uniquement les parties de traces qui représentent les comportements suspects de l'application ou du système tracé. Notre approche vise à produire, à partir de la trace d'origine, une trace réduite qui ne contient que des données pertinentes pour le débogage ou le profilage. Cette approche a le double objectif de réduire le volume de données à stocker et de présenter au développeur les informations les plus pertinentes sur l'exécution d'une application dans un contexte d'analyse.

Dans la suite de ce chapitre, nous présenterons à la section 4.2 les hypothèses et les définitions importantes associées à notre modèle de détection des régimes anormaux des applications. Dans la section 4.3, nous présenterons la phase d'apprentissage associée à notre approche Dans la section 4.4, nous présenterons la phase de détection des régimes anormaux.

4.2 Hypothèses et préliminaires

Pour élaborer notre approche de réduction des traces d'exécution, nous nous sommes intéressés aux traces d'exécution produites sur les systèmes embarqués. Sur ces architectures, les traces d'exécution sont la principale solution de débogage et de profilage des applications. Les traces d'exécution générées sur les systèmes embarqués sont constituées d'un mélange d'événements systèmes et d'événements applicatifs. Les événements systèmes constituent les opérations effectuées par le système d'exploitation. Comme événements de ce type, on distingue les interruptions ou les changements de contexte. Les événements applicatifs décrivent l'exécution de l'application observée tels que les appels de fonctions ou l'invocation d'une API.

Nos travaux ont été effectués au sein de l'entreprise STMicroelectronics. Au cours de ces travaux, nous avons remarqué que les applications les plus développées dans le milieu de l'embarqué sont les applications multimédia. Elles sont les plus critiques du point de vue commercial et possèdent la caractéristique d'être difficiles à déboguer ou profiler. C'est le cas des applications de décodage/encodage audio/vidéo implémentées pour les Set-Top Box, smartphones ou encore les téléviseurs 4K (UHD). Les outils d'analyse d'applications utilisés pour le débogage et le profilage de ces applications reposent principalement sur l'analyse des

traces d'exécution. Suivant ces observations, pour construire notre approche nous nous sommes focalisés sur les traces d'exécution produites par les applications multimédia sur les systèmes embarqués, notamment les Set-top box.

Un événement dans une trace d'exécution est décrit par un type d'événement et les paramètres qui lui sont associés. Le type d'événement renseigne sur la nature de l'événement (système ou applicatif) et l'action associée à l'événement (Entrée d'appel système, création de thread kernel, interruption, ...). Les paramètres spécifient les données et ressources utilisées pour la réalisation de l'action associée au type d'événement.

Exemple 1. *Dans la Figure 4.1, l'événement '__switch_to : tfm_86d7b680' est composé du type d'événement '__switch_to' qui désigne l'événement système 'changement de contexte' et du paramètre 'tfm_86d7b680' désigne le nom du processus auquel est attribué le processeur pour son exécution. De même, l'événement 'enter_user_fct : [stmcore_display_stiH416]' est composé du type d'événements 'enter_user_fct' qui désigne l'événement applicatif entrée fonction utilisateur pour lequel le paramètre associé 'stmcore_display_stiH416' désigne une fonction de l'application liée à l'affichage d'images.*

Dans notre approche et notre formalisation, nous ne considérerons pas les paramètres contenus dans les événements. Ils fournissent un niveau d'information très détaillé qui ne présente aucun intérêt pour notre modèle d'analyse des traces d'exécution. Nous nous intéresserons uniquement aux types d'événements contenus dans les traces d'exécution. Dans la suite, nous ferons un abus de langage entre type d'événement et événement en désignant par événement le type d'événement qui lui est associé.

De manière formelle, considérons l'ensemble Σ des événements associés à l'exécution d'une application. Un événement est dit horodaté s'il est associé à un timestamp. Il est représenté par une paire (t, e) où $t \in \mathbb{N}$ est un timestamp et $e \in \Sigma$ est un événement.

Définition 6 (Trace d'exécution)**.** *Une trace d'exécution T est une séquence d'événements horodatés ordonnés par timestamp : $T =< (t_0, e_0), (t_1, e_1), ..., (t_n, e_n) >$, $\forall i \in [0, n]$, $t_i \in \mathbb{N}$, $e_i \in \Sigma$, et $\forall i, j \in [0, n]$, si $i < j$ alors $t_i < t_j$.*

Exemple 2. *Un extrait de trace d'exécution présenté à la Figure 4.1 se représente par la séquence suivante : <..., (1863107917414988, enter_interrupt:mailbox), (1863107917415000, exit_interrupt:), (1863107917415003, __switch_to:tfm_86d7b680), (1863107917415024, __switch_to:SE-Audio-Mixer), (1863107917415025, __switch_to:0), ... >*

Selon Vijayakumar et al. dans [VN12], le fonctionnement des applications multimédia est associé à la notion de *frames*, où une frame représente une instance de sémantique métier. Dans les frames, les mêmes tâches sont presque toujours réalisées, notamment l'initialisation, l'acquisition des données, le décodage audio, le décodage vidéo, etc. Les frames se traduisent dans les traces d'exécution par des répétitions de certains ensembles d'événements, ce qui permet d'associer un comportement décrit par un ensemble répétitif d'événements à une sémantique métier de l'application. Une illustration d'ensemble répétitifs d'événements dans une trace d'exécution de décodage vidéo est présentée à la Figure 4.2.

Suivant cette hypothèse, nous considérons les ensembles d'événements répétitifs, contenus dans la trace d'exécution, comme le modèle de base pour la description des régimes de fonctionnement d'une application. Pour identifier ces régimes, il est nécessaire d'analyser toute la trace d'exécution. Toutefois, nous ne

```
...
1863107917414988        enter_interrupt: mailbox
1863107917415000        exit_interrupt:
1863107917415003        __switch_to: tfm_86d7b680
1863107917415024        __switch_to: SE-Audio-Mixer
1863107917415025        __switch_to: 0
1863107917415049        __switch_to: 0
1863107917420869        __switch_to: SE-Audio-DtoM
1863107917420876        __switch_to: SE-ManifestCoor
1863107917420890        __switch_to: SMF-Aud
1863107917420893        exit_sys_call: sys_poll
1863107917420893        __switch_to: 0
1863107917420903        __switch_to: 0
1863107917420930        __switch_to: SE-Video-MbufR
1863107917420937        __switch_to: SE-Video-DtoM
1863107917420942        __switch_to: SMF-Aud
1863107917420947        __switch_to: multiqueue0:src
1863107917420955        enter_sys_call: sys_ioctl
1863107917420955        __switch_to: 0
1863107917420965        exit_sys_call: sys_ioctl
1863107917420974        __switch_to: 0
1863107917425817        enter_interrupt: vsync0
1863107917425878        exit_interrupt:
1863107917425884        enter_interrupt: displaylink
1863107917425886        enter_user_fct: [stmcore_display_stiH416]
1863107917425888        enter_user_fct: [stmcore_display_stiH416]
1863107917425897        exit_user_fct: [stmcore_display_stiH416]
1863107917425898        exit_user_fct: [stmcore_display_stiH416]
1863107917425900        exit_interrupt:
1863107917425904        enter_interrupt: displaylink
...
```

FIGURE 4.1 – Exemple de trace : les événements systèmes sont de couleur noire et les événements applicatifs de couleur rouge

voulons pas stocker la totalité de la trace avant de l'analyser. Nous souhaitons analyser la trace d'exécution au fur et à mesure qu'elle est générée et ne voulons faire aucune hypothèse sur la taille totale de la trace d'exécution. Elle peut ne pas tenir en mémoire car elle est potentiellement infinie. Pour faire face à ces contraintes, nous nous proposons d'utiliser une technique d'approximation de l'historique sur les données, *la fenêtre glissante* [GZK05]. Le principe de la fenêtre glissante consiste à focaliser l'analyse sur les instances de données les plus récentes tout en considérant un résumé global des anciennes instances. Dans notre approche, nous intégrons le modèle de fenêtre glissante afin de découper la trace d'exécution en sections d'événements. Dans le but d'élaborer un modèle de données simple à manipuler, nous n'autoriserons pas les chevauchements entre les fenêtres.

Définition 7 (Fenêtre d'événements)**.** *Etant donnée une date de début de fenêtre s et une période p, une fenêtre d'événement $W_{s,p}$ est une sous-séquence d'événements horodatés*

$$w_{s,p} =< (t_s, e_s), (t_{s+1}, e_{s+1}), ..., (t_{s+p}, e_{s+p}) >$$

avec $t_0 \leq t_s \leq t_{s+p} \leq t_{0+n}$ *et* $0 \leq s \leq n - p$

Suivant ce modèle d'analyse, une trace d'exécution apparaît comme une séquence de fenêtres d'événements disjointes et contiguës. Chaque fenêtre de la trace d'exécution d'une application représente une observation de l'état d'exécution de l'application.

Exemple 3. *Sur la Figure 4.3, nous présentons un exemple de découpage d'une portion de trace en deux fenêtres temporelles consécutives. Chaque fenêtre d'événements dure 10 ms.*

FIGURE 4.2 – Illustration des frames dans une trace. $D = \{$*enter_user_fct:player2, __switch_to:multiqueue0:src, enter_interrupt:h264pp_0, __switch_to:SE-VideoH264-In, enter_user_fct:stmcore_display_stiH416, __switch_to:SE-Audio-Mixer*$\}$ est un exemple d'ensemble d'événements communs aux frames 1 et 2. Les éléments de D, présentés en couleur bleue dans la trace, représentent les événements liés au décodage audio-vidéo d'un flux vidéo.

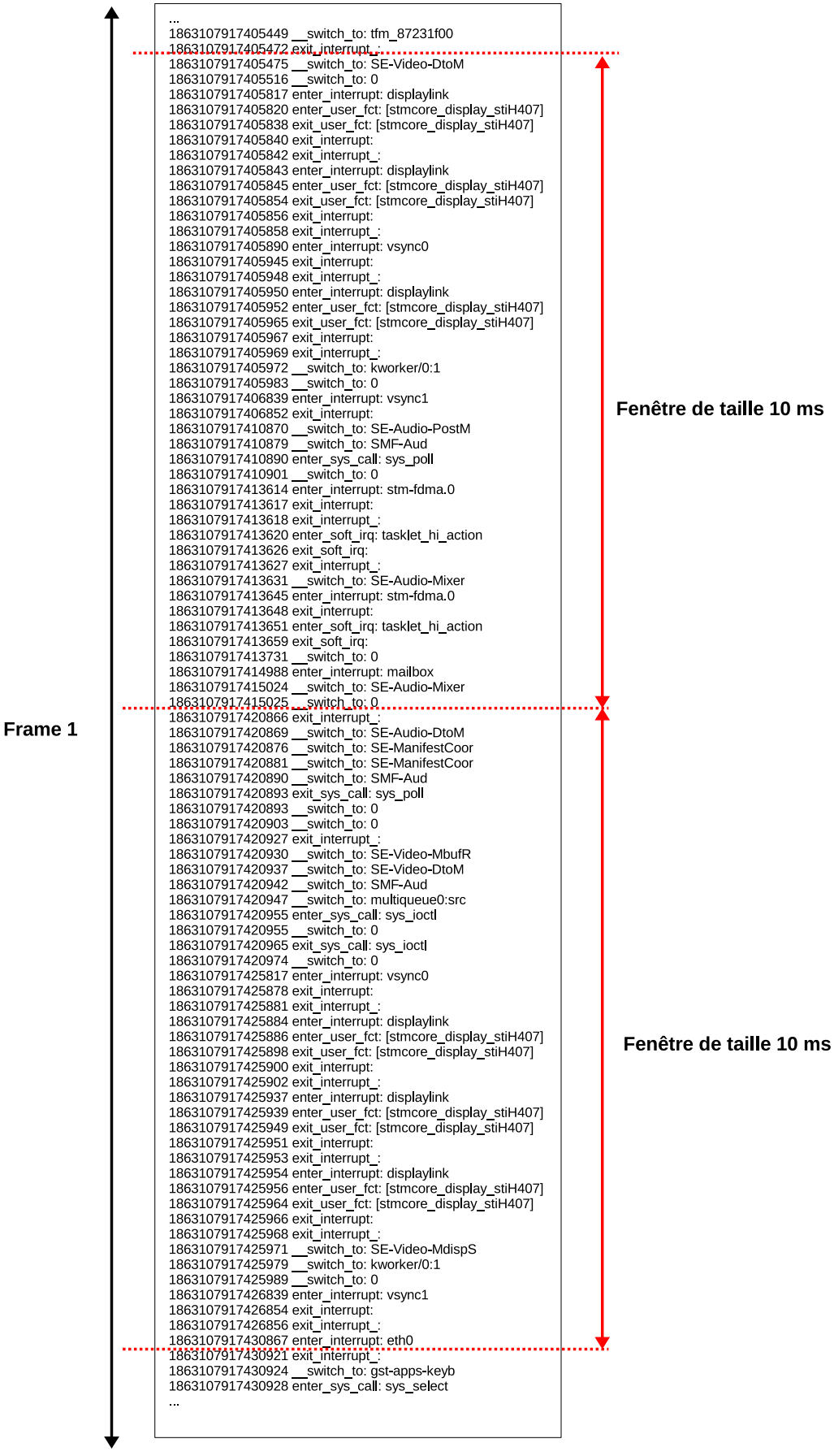


FIGURE 4.3 – Découpage de la trace en fenêtres temporelles de 10 ms

Le principe de l'analyse d'une trace d'exécution repose sur la capture du régime de fonctionnement de l'application associé à chaque section de la trace d'exécution. Suivant la taille d'une fenêtre, la section de la trace associée peut modéliser une partie d'un régime de fonctionnement (grain fin), un régime entier de fonctionnement ou une agrégation de plusieurs régimes de fonctionnement (gros grain). Etant donné que nous ne disposons généralement d'aucune connaissance a priori sur la durée des régimes de fonctionnement des applications et que de plus cette durée peut varier en fonction de plusieurs paramètres tels que la nature du régime, l'application ou encore l'architecture d'exécution, nous choisissons d'utiliser une taille de fenêtre qui correspond à un découpage à grain fin. Ce choix est guidé par l'intuition que le découpage à grain fin assure une restitution fine des différents régimes contenus dans la trace d'exécution alors qu'un découpage à gros grain agrégerait plusieurs régimes avec le risque de masquer ceux qui représenteraient des comportements anormaux de l'application tracée.

Pour analyser le fonctionnement de l'application à partir de sa trace d'exécution, il est nécessaire de décrire les régimes associés aux fenêtres contenues dans la trace. Pour cela, nous avons besoin d'un modèle adéquat pour la description d'un régime. Trois modèles se dégagent :

- modèle 1 : les ensembles d'événements. Ce modèle repose sur l'hypothèse selon laquelle les régimes similaires présentent les mêmes ensembles d'événements (cas du décodage de frames vidéos à la Figure 4.2). Suivant ce modèle, un régime associé à une fenêtre peut être décrit par un ensemble d'événements contenus dans la fenêtre.
- modèle 2 : les distributions d'événements. Ce modèle décrit le régime associé à une fenêtre d'événements par un résumé de l'information contenue dans la fenêtre. Ce résumé se représente par la distribution des événements contenus dans la fenêtre [KBDG04, PPB+06]. Ce modèle considère les différents événements d'une fenêtre comme des variables aléatoires dont le nombre d'occurences dans la fenêtre est associé à une loi de probabilité [DGIM02b].
- modèle 3 : les séquences d'événements (avec ou sans répétition des événements). Ce modèle décrit le régime associé à une fenêtre par une séquence où les événements de la fenêtre constituent les items de la séquence et l'ordre d'occurrences des événements est le même que celui des items dans la séquence. Ce modèle de description des régimes permet de prendre en compte les dépendances entres les événements à travers leur ordre d'occurrence .

Parmi ces modèles, nous choisissons d'utiliser le modèle 2, qui décrit un régime par la distribution d'événements contenus dans la fenêtre associée. Le modèle 1, pour lequel les régimes sont décrits par les ensembles d'événements, est insuffisant pour décrire les régimes associés aux fenêtres en ce sens que deux fenêtres de régimes différents peuvent contenir les mêmes ensembles d'événements. Le modèle 3, qui décrit le régime associé à une fenêtre par sa séquence des événements, repose sur l'ordre d'occurrence des événements. La prise en compte de l'ordre d'occurrence des événements constitue une difficulté pour l'analyse des traces d'exécution. Cette difficulté est liée au non-déterminisme induit par l'ordonnanceur de l'environnement d'exécution. Deux instances d'exécution similaires d'une même application sur un même jeu de données et sur un même environnement d'exécution peuvent produire deux traces d'exécution différentes au niveau de l'ordre des occurrences d'événements [SW07]. Le modèle 2 se présente comme un modèle plus complet que le modèle 1 pour la description des régimes associés aux fenêtres d'événements étant donné qu'il prend en compte les occurrences des événements dans les fenêtres. De plus, ce modèle fait abstraction de l'ordre d'occurrence des événements dans les fenêtres, ce qui permet d'éviter les difficultés liées au non déterminisme de l'exécution.

Outre le fait que les distributions d'événements permettent de faire abstraction de l'ordre des événements, Kifer montre dans [KBDG04] qu'une distribution d'événements représente un résumé de l'information contenue dans une fenêtre d'événements. Suivant la même idée, Dasu et al. dans [DKVY06] utilisent le modèle de distribution pour exprimer la quantité d'information associée à une fenêtre d'événements afin d'utiliser les outils comme l'information mutuelle pour comparer les fenêtres d'événements.

Dans notre approche, nous utiliserons la notion de distribution empirique des événements décrite sous la forme d'un histogramme pour représenter le régime associé à une fenêtre d'événement [PPB+06].

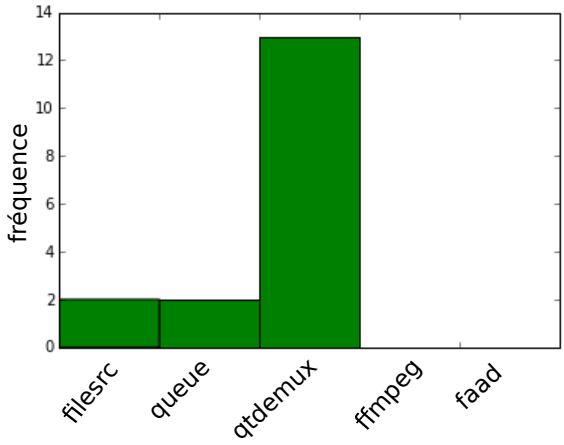
```
...
1629503799 ffmpeg   'clipped timestamp:0:00:01.560000000'
1629519590 ffmpeg   'not dropping'
1629533967 ffmpeg   'return flow 0'
1629548865 ffmpeg   'Decoded data'
1645686167 queue    'last_stop updated'
1645734352 queue    'sink 0:00:02.600634920'
1645781346 faad     'buffer of size 519'
1646050022 faad     '519 bytes consumed'
1646136412 queue    'last_stop updated to 0:00:01.904036281'
1646156514 queue    'sink 0:00:02.600634920'
1646175541 faad     'buffer of size 506'
1646397455 faad     '506 bytes consumed'
1667083138 queue    'sink 0:00:02.600634920'
1667105309  faad    'buffer of size 521'
1671712134 ffmpeg   ''
1671755582 queue    'last_stop updated to 0:00:01.640000000'
1672316124 qtdemux 'pushing from stream'
1672352434 qtdemux 'reading 476 bytes @ 5896775'
1672388835  filesrc   'Reading 476 bytes at offset 0x59fa47'
1672427082 qtdemux 'Pushing buffer with time 0:00:02.600634920'
1672461464 queue    'last_stop updated to 0:00:02.623854875'
1672488754 queue    'sink 0:00:02.623854875'
...
```

FIGURE 4.4 – Extrait d'une trace d'exécution

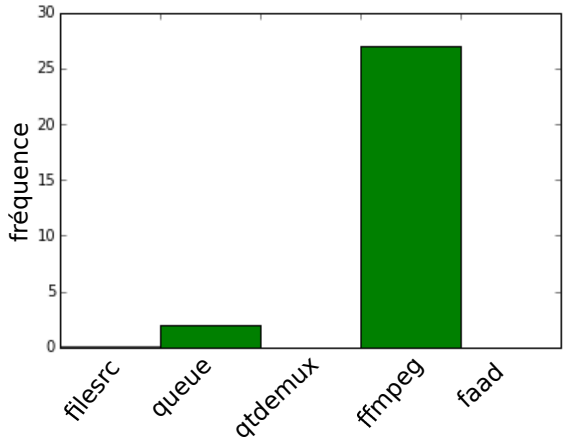
Exemple 4. *Une analyse effectuée sur une trace d'exécution (extrait illustré à la Figure 4.4) d'une application de décodage audio/vidéo basée sur le framework Gstreamer présente des distributions d'événements qui décrivent les principales étapes de traitements de l'application. Sur cette trace, les événements capturés sont uniquement ceux liés aux composants impliqués dans le processus de décodage audio/vidéo, notamment le composant « filesrc » pour la lecture des données binaire, le composant « queue » pour la gestion des files de communication, le composant « demux » pour la démultiplexage des données audio/vidéo, le composant « ffmpeg » pour le traitement des données vidéo et le composant « faad » pour le traitement des données audio. Les distributions identifiées permettent de décrire les régimes liés à la lecture de données, au démultiplexage des données audio/vidéo, au décodage des données audio et au décodage des données vidéo à partir de leur distribution d'événements. Des exemples de distributions sont présentés en Figure 4.5.*

Pour calculer en toute rigueur la distribution empirique des items associée à une séquence, il est nécessaire de transtyper cette séquence en un multiensemble où chaque item de la séquence constitue un item du multi-ensemble [DKVY06].

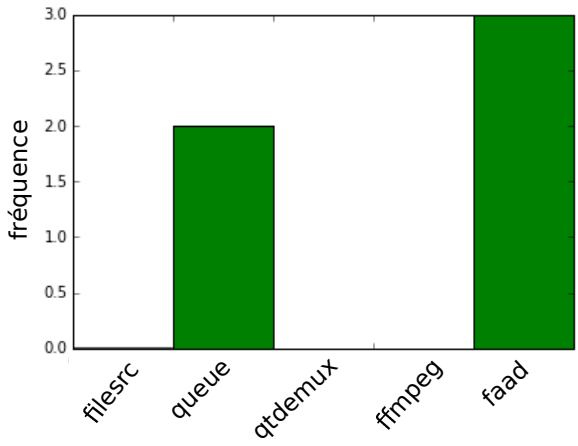
Définition 8 (Distribution d'événements associée à une séquence)**.** *Soit une fenêtre d'événements* $w =< (t_1, e_1), (t_2, e_2), ..., (t_m, e_m) >$*, et* $m_w = \{e_1, e_2, ..., e_m\}$ *le multi-ensemble associé à* w *dans l'alphabet fini* Σ*. La distribution des événements de* w *est le vecteur* P_w *associé à la fréquence relative de chaque*



(a) Distribution d'événements liés à lecture des données



(b) Distribution d'événements liés au décodage vidéo



(c) Distribution d'événements liés au décodage audio

FIGURE 4.5 – Exemple des distributions d'événements associés aux fonctionnalités métiers d'une application de décodage audio/vidéo

élément de Σ *dans* w

$$P_w(e_i) = \frac{N(e_i|m_w)}{n}$$

où $N(e_i|m_w)$ *est le nombre d'occurrence de l'événement* e_i *dans* m_w *et* $n = |m_w|$.

Exemple 5. *Considérons la trace présentée à la Figure 4.6, avec* $\Sigma = \{a, b, c, d\}$. *Cette trace est fragmentée en deux fenêtres* w_1 *et* w_2. *Les multi-ensembles associés à* w_1 *et* w_2 *sont respectivement* $m_{w_1} = \{a, a, a, b, c\}$ *et* $m_{w_2} = \{a, b, b, b\}$. *Les distributions d'événements associées à* w_1 *et* w_2 *sont les histogrammes* h_{w_1} *et* h_{w_2} *représentant les fréquences des éléments de* w_1 *et* w_2 *avec* $hist_{w_1} = \{p_{w_1}(a), p_{w_1}(b), p_{w_1}(c), p_{w_1}(d)\}$ *où* $p_{w_1}(a) = 3/5$, $p_{w_1}(b) = 1/5$, $p_{w_1}(c) = 1/5$ *et* $p_{w_1}(d) = 0$, *et* $hist_{w_2} = \{p_{w_2}(a), p_{w_2}(b), p_{w_2}(c), p_{w_2}(d)\}$ *où* $p_{w_2}(a) = 1/4$, $p_{w_2}(b) = 3/4$, $p_{w_2}(c) = 0$ *et* $p_{w_2}(d) = 0$.

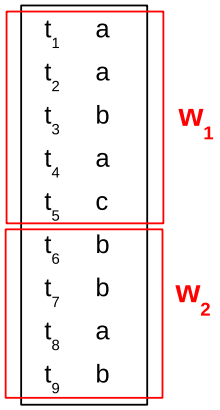


FIGURE 4.6 – Exemple de trace fragmentée en fenêtres d'événements

4.3 L'apprentissage automatique des régimes réguliers

Au cours de son exécution, une application va répéter périodiquement l'exécution de certains traitements sur les données (d'entrée ou intermédiaires). Cette répétition va s'instancier dans la trace d'exécution par une répétition de certains régimes de fonctionnement de l'application. Nous qualifions de *régimes réguliers* ou *régimes répétitifs* les régimes qui se produisent fréquemment au cours de l'exécution de l'application. On suppose que ces régimes ne sont pas connus à l'avance, ce qui simplifie l'utilisation de notre approche.

Notre approche d'identification des régimes réguliers d'une application repose sur l'intuition que les fenêtres d'événements associées à un même régime de fonctionnement présentent des distributions similaires d'événements, autrement dit ces fenêtres d'événements partagent une information mutuelle importante. Par contre, les fenêtres associées à des régimes de fonctionnement différents présentent peu de similarité entre leur distribution d'événements et donc partagent peu d'information mutuelle.

La phase d'apprentissage automatique des régimes réguliers consiste donc à identifier parmi les régimes d'une application ceux qui sont réguliers et ce, sans aucune connaissance a priori. Mesurer la similarité entre deux distributions de données revient à exprimer une notion de distance entre ces distributions. Un outil permettant de mieux exprimer cette notion de distance est l'entropie relative de la théorie de l'information encore appelée *divergence de Kullback-Leibler* [DKVY06]. La divergence de Kullback-Leibler est

un outil de la théorie de l'information qui permet d'évaluer la quantité d'information mutuelle entre deux distributions de probabilité. Si deux distributions de probabilité présentent une information mutuelle importante (une valeur de divergence proche de 0), alors ces distributions sont similaires. Par contre si deux distributions de probabilité présentent une valeur de divergence élevée alors, ces distributions partagent très peu d'information mutuelle et de ce fait ne sont pas similaires.

Définition 9 (La divergence de Kullback-Leibler)**.** *Soit deux distributions discrètes p et q. La divergence Kullback-Leibler ou (KL-divergence) [KL51] entre p et q se définit comme :*

$$D_{KL}(p,q) = \sum_{e \in \Sigma} p(e) \cdot log \frac{p(e)}{q(e)}$$

où Σ est l'univers des événements associés à la trace d'exécution.

La KL-divergence s'apparente à la notion de distance mais cependant ne respecte pas toutes les propriétés d'une distance, notamment la symétrie, $D_{KL}(p,q) \neq D_{KL}(q,p)$, et l'inégalité triangulaire. Dans notre cas, cela voudrait dire qu'il peut arriver que deux distributions p et q soient similaires parce que $D_{KL}(p,q)$ est très proche de 0, mais par contre que q et p ne soient pas similaires parce que $D_{KL}(q,p)$ est grand. Pour des raisons de stabilité de notre modèle d'évaluation de la régularité des distributions, il est important que la mesure de similarité respecte la propriété de symétrie. Pour ce faire, nous utiliserons la *divergence de Jeffreys* qui est une variante de la KL-divergence et qui prend en compte la propriété de symétrie [Evr13].

Définition 10 (La divergence de Jeffreys)**.** *Soit deux distributions discrètes p et q. La divergence de Jeffreys [Jef61] entre p et q se définit comme :*

$$D_J(p,q) = \sum_{e \in \Sigma} (p(e) - q(e)) \cdot log \frac{p(e)}{q(e)}$$

où Σ est l'univers des événements associés à la trace d'exécution.

Pour réaliser l'apprentissage automatique, il est nécessaire de disposer d'un ensemble de distributions d'événements à partir duquel les régimes réguliers seront identifiés. Cet ensemble de distributions d'événements peut être construit de deux façons :

- soit on dispose d'une ou plusieurs traces d'exécution issues de l'exécution de l'application analysée comportant aucun ou très peu d'états de fonctionnement anormaux,
- soit on suppose qu'une partie de l'exécution de l'application est quasi-correcte, c'est-à-dire avec très peu de régimes irréguliers. Cette partie de l'exécution peut être observée et facilement validée par le développeur comme trace de référence.

Le deuxième cas d'usage sied mieux au contexte de validation d'application ou de tests d'endurance pour lesquels on est sûr que l'application testée fonctionne bien sur des petites périodes d'exécution, périodes pour lesquelles il est facile de réaliser une vérification graphique. L'objectif de la validation d'applications ou des tests d'endurance d'une application consiste à étudier son comportement sur une longue période d'exécution.

À partir de cet ensemble de distributions d'événements et suivant l'hypothèse sur les régimes réguliers, nous nous proposons d'identifier les régimes réguliers en regroupant les distributions d'événements en fonction de leur similarité.

Notre but est d'apprendre les distributions d'événements caractéristiques des régimes réguliers. Pour cela, nous comparons les distributions associées aux fenêtres d'événements constituant la trace d'exécution afin d'identifier celles qui présentent des similarités. Ces distributions sont considérées comme les instances d'un régime de fonctionnement, et de ce fait, caractérisent la répétition d'un régime.

Pour identifier automatiquement les états d'exécution répétitifs, nous allons représenter les distributions extraites des fenêtres d'événements sous forme de points dans un espace de dimension $|\Sigma|$ où chaque dimension est associée à un type d'événement. La valeur de la projection d'un point sur une dimension représente le nombre d'occurrences de l'événement associé dans la fenêtre. Suite à la représentation des distributions de l'ensemble d'apprentissage dans l'espace $|\Sigma|$-dimensionnel, nous utilisons l'approche LOF pour identifier les points qui constituent des zones denses (clusters) et les points qui se situent dans les zones à faible densité. Les points qui forment un cluster représentent des distributions similaires et les points situés à l'extérieur des clusters c'est-à-dire dans les zones à faible densité représentent les distributions non-répétitives.

Nous considérons les régions denses identifiées comme des clusters de références. En effet, ces régions denses sont formées par des points qui représentent les distributions d'événements. régions s'assimilent à des régimes d'exécution pour lesquels l'application présente des comportements réguliers (normaux).

Les distributions qui forment les clusters de références sont utilisées à l'étape suivante, l'étape de détection, dans le but d'identifier les régimes suspects. Pour cela, les distributions qui appartiennent aux zones non-denses sont supprimées des données d'apprentissage pour éviter leur implication dans la phase de détection, car ils réduiraient la qualité de détection des régimes irréguliers.

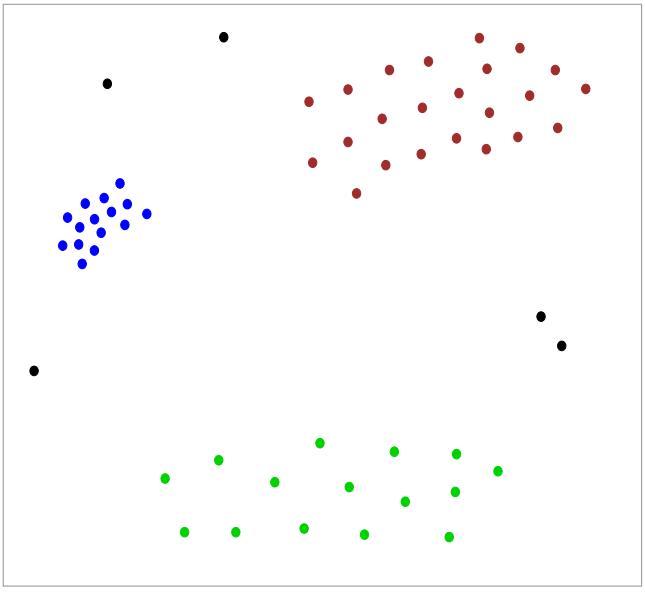


FIGURE 4.7 – Illustration des régimes réguliers dans un espace à deux dimensions : 3 clusters de points apparaissent clairement, ils constituent les clusters de référence et représentent les régimes réguliers. Les points de couleur noire n'appartiennent à aucun cluster, ils représentent les régimes irréguliers

4.4 La phase de détection de régimes irréguliers

La phase de détection des régimes irréguliers constitue la seconde phase de notre approche. Elle est effectuée après la phase d'apprentissage automatique. Durant cette phase, chaque fois qu'une nouvelle fenêtre d'événement est disponible, sa distribution d'événement est comparée avec les distributions présentes dans les clusters de référence. Si la distribution de cette fenêtre est très similaire aux distributions contenues dans un cluster alors la fenêtre d'événements représente un régime régulier. Par contre si la distribution de la nouvelle fenêtre présente une faible similarité avec les distributions des clusters de référence alors la fenêtre d'événements représente un régime irrégulier et de ce fait, elle représente un *comportement suspect*.

Une approche naïve consiste à comparer la distribution d'une fenêtre de la phase de détection avec toutes les distributions des clusters de référence pour déterminer si la distribution appartient ou non à un cluster de référence. Cependant une telle approche est coûteuse en calculs, notamment lorsque le nombre de distributions extraites pendant la période d'apprentissage est important et que ces distributions se regroupent en des clusters de forme arbitraire.

Pour comparer la distribution d'une fenêtre avec les distributions des clusters de référence, nous utilisons la méthode LOF qui compare une distribution d uniquement avec les distributions des clusters de référence qui lui sont proches. Ainsi, si une distribution présente les mêmes caractéristiques de densité que les distributions dont elle est proche, alors cette distribution appartient au cluster de référence de ses distributions voisines, et de ce fait modélise un régime régulier. Sinon cette distribution est une anomalie et représente un régime irrégulier.

Exemple 6. *Sur la Figure 4.8, nous présentons les clusters de référence (groupes de points de couleur bleue, verte et marron) que forment les distributions des fenêtres extraites de la trace d'exécution pendant la phase d'apprentissage. Nous y présentons aussi quatre points p_1, p_2, p_3 et p_4 représentant les distributions associées respectivement à quatre fenêtres w_1, w_2, w_3 et w_4 de la trace pendant la phase détection. Chacun des points p_1, p_2, p_3 et p_4 est évalué par rapport à ses trois voisins les plus proches. Un score d'anomalie est calculé pour chacun des points suivant l'approche LOF, et nous remarquons dans ce cas que les points (p_1 et p_3) dont les scores d'anomalie sont inférieurs à 2 se situent dans les clusters de référence et de ce fait les fenêtres d'événements (w_1 et w_3) associées représentent des régimes réguliers. Par contre les points (p_2 et p_4) dont les scores d'anomalie sont supérieurs à 2 se situent à l'extérieur des clusters, les fenêtres d'événements w_2 et w_4 représentent des régimes irréguliers et donc suspects.*

La phase de détection des régimes irréguliers est réalisée à la volée. Chaque fois qu'une fenêtre d'événements, ou qu'un batch de fenêtres d'événements est disponible, les distributions d'événements associées sont comparées avec les distributions des clusters de référence. Les distributions des clusters de référence choisis pour la comparaison sont celles qui présentent le plus de similarité avec les distributions pour lesquelles on évalue le caractère régulier ou non. Lorsqu'une distribution modélise un régime irrégulier, la section de trace d'exécution associée à cette distribution est automatiquement stockée.

L'approche LOF repose sur la sélection des plus proches voisins pour évaluer la densité autour d'un point en terme de proximité avec ses plus proches voisins. Dans l'approche LOF, la complexité en calcul est dominée par le calcul de la densité (du score d'anomalie) autour de chaque point. La complexité en calcul de l'approche LOF est de l'ordre de $O(nlogn)$ où n est le nombre de points. Le calcul du voisinage

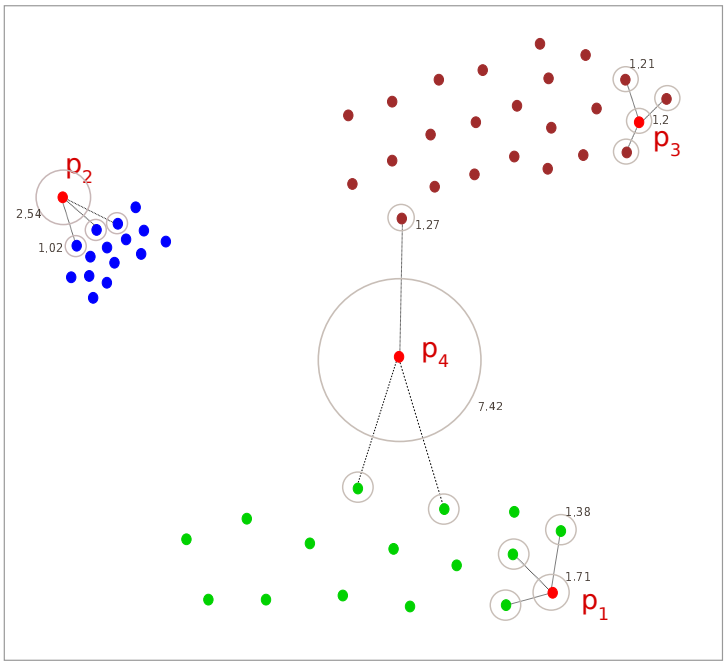


FIGURE 4.8 – Illustration de la détection des régimes réguliers et irréguliers pendant la phase de détection : les points p_1 et p_3 dont les scores d'anomalies sont respectivement 1.71 et 1.2 se situent respectivement dans les clusters de points verts et marrons, et représentent les régimes réguliers. Les points p_2 et p_4 de scores d'anomalie respectifs 2.54 et 7.42 sont distants des clusters, ils représentent des régimes irréguliers.

est peu coûteux comparé au calcul de la densité d'un point.

Pour éviter le calcul du score d'anomalies (valeur de LOF) pour toutes les fenêtres qui constituent la trace d'exécution, nous réalisons une comparaison préliminaire sur les fenêtres consécutives basées sur leur quantité d'information mutuelle. Si ces deux fenêtres consécutives présentent une quantité d'information mutuelle importante, alors on considère que ces deux fenêtres représentent un même régime de fonctionnement et peuvent être agrégées en une seule fenêtre. Le score d'anomalie ne sera alors calculé que pour la fenêtre agrégée et non pour deux fenêtres.

Le processus de détection des régimes irréguliers est décrit dans l'*Algorithme 1*. Le principe de l'algorithme est le suivant : l'analyse de la trace d'exécution est réalisée à la volée, les données sont analysées dès qu'elles sont produites. Ainsi, dès qu'une fenêtre d'événements w_{curr} est disponible (ligne 5), la distribution des événements associée d_{curr} est calculée (ligne 6), puis comparée avec la distribution de la fenêtre d'événements précédente d_{prev} (ligne 7), dans le cas d_{curr} et d_{prev} sont similaires, alors les fenêtres d'événements respectives sont agrégées en une seule. Dans le cas contraire, c'est à dire le cas où d_{curr} et d_{prev} ne sont pas similaires, d_{prev} est comparée avec les distributions de référence et un score d'anomalie est calculé (ligne 11). Si le score d'anomalie est supérieur au seuil d'anomalie (ligne 12) alors la fenêtre d'événement w_{prev} associée à d_{prev} est produite en sortie pour être stockée. Les données produites en sortie par l'*Algorithme 1* sont les fenêtres d'événements associées aux comportements irréguliers de l'application dont la trace est analysée.

Nous venons de décrire l'approche de réduction des traces d'exécution que nous proposons. Cette approche utilise la méthode LOF pour réaliser la détection à la volée des régimes irréguliers contenus dans

Algorithm 1: Détection des régimes irréguliers

input :

T : la trace,

k : le nombre de voisins proches pour le calcul de la LOF,

l : la taille d'une fenêtre,

C : l'ensemble des clusters de référence,

$lofThreshold$: le seuil d'anomalie

```
1  w_prev ← ∅;
2  d_prev ← 0;
3  while true do
4  |  // Extraire une section de trace suivant la taille de fenêtre
   |     d'événement
5  |  w_curr ← AcquireEventWindow(T, l);
6  |  d_curr ← ComputeEventDistribution(w_curr);
7  |  if Similarity(d_prev, d_curr) < minSim then
8  |  |  w_prev ← AggregateWindows(w_prev, w_curr);
9  |  |  d_prev ← AggregateDistributions(d_prev, d_curr);
10 |  else
11 |  |  LOF ← LOF(k, C, d_prev);
12 |  |  if LOF > lofThreshold then
13 |  |  |  // La fenêtre w_prev représente un régime irrégulier
14 |  |  |  output w_prev
15 |  |  end
16 |  |  d_prev ← d_curr ;
17 |  |  w_prev ← w_curr ;
18 |  end
19 end
```

une trace d'exécution. Afin d'évaluer l'effectivité de détection et de réduction de notre approche, nous présentons les expérimentations effectuées et les résultats obtenus au prochain chapitre.

CHAPITRE 5

Expériences et évaluations

Sommaire

Afin d'appliquer notre approche de réduction de traces sur les données réelles, nous avons implémenté un prototype à partir duquel nous avons mené une expérimentation détaillée. Cette expérimentation a pour objectif d'évaluer les aspects pertinents liés à notre approche de réduction notamment la qualité de détection des régimes irréguliers, le passage à l'échelle, les performances de réduction ou encore la sensibilité aux paramètres d'analyse. Pour ce faire, des expériences ont été effectuées sur des données synthétiques, d'une part, et sur des données réelles provenant d'applications multimédia et des systèmes embarqués STMicroelectronics d'autre part. Ce chapitre présente dans la section 5.1 les expériences réalisées et les résultats obtenus sur des données synthétiques. Par la suite, nous présentons à la section 5.2, les expériences réalisées et les résultats obtenus sur les données réelles. La section 5.3 conclut le chapitre à travers un résumé sur les différents résultats des expérimentations.

5.1 Expérimentations sur les données synthétiques

Dans cette section, nous présentons les expériences réalisées sur des données synthétiques. Le but de ces expériences est d'estimer les performances de détection des anomalies par notre approche. Les données synthétiques seront donc élaborées de manière à exprimer une forte régularité d'une part et d'autre part à contenir quelques instances irrégulières représentant des anomalies. L'élaboration des données synthétiques est réalisée en faisant varier les paramètres de génération des données, ceci afin d'évaluer la robustesse de notre approche. Pour évaluer la qualité de détection, nous utiliserons les mesures telles que la *précision*, le *rappel*, la courbe *sensibilité/spécificité* ou courbe *ROC*.

5.1.1 Description des données synthétiques et mesures d'évaluation

Description des données synthétiques

Les jeux de données synthétiques élaborés dans le cadre de nos expérimentations présentent une structure proche de celle des traces d'exécution réelles. Leurs données sont organisées en lignes. Chaque ligne est constituée d'un timestamp et d'une information qui représente un type d'événements. Cette information est encodée par un entier. Une illustration est proposée à la Figure 5.1.

```
...
100000    29
100100    32
100200    30
100300    28
100400    25
100500    26
100600    27
100700    24
100800    24
100900    31
101000    14
101100    15
101200    12
101300    10
101400    18
...
```

FIGURE 5.1 – Extrait d'un jeu de données synthétiques : La première colonne représente les timestamps et la seconde colonne, des types d'événements associés à chaque timestamp

Le principe de construction des données synthétiques repose sur la génération de fenêtres d'événements dont une grande proportion correspond à différents régimes de régularité et le reste correspond à des régimes différents des régimes de régularité. Dans notre méthode de génération de données, les régimes sont associés à des distributions d'événements. Cette méthode de génération des données synthétiques est présentée dans l'Algorithme 2.

Notre méthode de génération utilise une liste d'entiers $eventTypeList$ fournie en entrée qui définit les différents types d'événements dans les données synthétiques. Chaque entier de $eventTypeList$ désigne un type d'événement.

La première étape consiste à construire les distributions d'événements à partir desquelles les jeux de données synthétiques seront générés. La construction des distributions d'événements est faite par des tirages aléatoires successifs d'événements dans l'ensemble des types d'événements ($eventTypeList$) (Ligne 2). Lors de la construction des distributions d'événements, nous fixons une valeur minimale de divergence ($divergencethreshold$) (Ligne 2) pour assurer une différence (divergence) minimale entre les distributions d'événements, ceci afin de garantir leur diversité. Cette diversité a pour but de simuler une certaine variété des régimes telle que celle observée sur les jeux de données réelles. Par la suite, les distributions d'événements construites sont séparées en deux groupes : le groupe des distributions étiquetées régulières qui permettra de générer les fenêtres d'événements considérées comme régulières (Ligne 3), et le groupe des distributions étiquetées irrégulières qui permettra de générer les fenêtres d'événements considérées irrégulières (Ligne 4).

Algorithm 2: Synthetic data generator

input :

$eventTypeList$: liste des types d'événements,
n : nombre de distributions,
$nRegD$: nombre de distributions régulières,
$maxLearnSize$: nombre max des fenêtres pour l'apprentissage,
$nWindows$: nombre de fenêtres à générer,
$divergencethreshold$: seuil de divergence
$noisethreshold$: seuil de bruit dans la génération des données
$tsRange$: plage de timestamps associés aux événements d'une fenêtre

output:

$\mathcal{D}$: liste des fenêtres d'événements horodatés.

$\mathcal{D} \leftarrow \emptyset$;
$\mathcal{S} \leftarrow$ `generateDistributions(`$eventTypeList$, n, $divergencethreshold$`)`;
$RegularDistributions \leftarrow \mathcal{S}[1 : nRegD]$;`// Sélection des distributions régulières`
$IrregularDistributions \leftarrow \mathcal{S} - RegularDistributions$;`// Sélection des distributions irrégulières`
for $i = 0$ *to maxLearnSize* **do**
  $distrib \leftarrow$ `getDistribution(`$RegularDistributions$`)`;
  $window \leftarrow$ `generateTimestampedWindow(`$tsRange$, $noisethreshold$, $distrib$`)`;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{window\}$;
end
for $i = maxLearnSize$ *to* $nWindows$ **do**
  $distrib$ = `getDistribution(`$\mathcal{S}$`)`;
  $window \leftarrow$ `generateTimestampedWindow(`*tsRange*, $noisethreshold$, $distrib$`)`;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{window\}$;
end
output $\mathcal{D}$

La seconde étape (Ligne 6 à 7) consiste à générer des fenêtres d'événements à partir des distributions construites à l'étape précédente. À chaque occurrence d'un événement dans une fenêtre d'événements est associée un timestamp. Les timestamps sont générés suivant la loi uniforme et doivent être compris dans la plage de timestamps $tsRange$.

Pour rapprocher le schéma des données synthétiques de celui des données réelles, nous introduisons un bruit contrôlé dans la construction des fenêtres d'événements, précisément sur les nombres d'occurrences des événements. Ce bruit permet de modéliser dans les données synthétiques les écarts (variations) qui existent entre les distributions d'événements modélisant un même régime de fonctionnement de l'application. Outre cette similitude avec les jeux de données réels, l'introduction d'un bruit contrôlé dans les jeux de données synthétiques nous permettra d'estimer la sensibilité du module de détection d'anomalies de notre approche en fonction du seuil de bruit dans les données. À cet effet, considérons $n_{D_j}(e_i)$ le nombre d'occurrences de l'événement e_i dans la distribution D_j, insérer un bruit de $d\%$ sur une fenêtre d'événements générée à partir à D_j consiste à produire pour tout événement e_i dans D_j, un nombre d'occurrences $n'_{D_j}(e_i)$ tels que $n_{D_j}(e_i) - d\% \leq n'_{D_j}(e_i) \leq n_{D_j}(e_i) + d\%$.

Chaque jeu de données généré par notre algorithme de génération de données synthétiques est constitué de deux parties : une partie constituée uniquement des fenêtres d'événements étiquetées régulières (Lignes 5 à 9) et une partie constituée des fenêtres étiquetées régulières et irrégulières (Lignes 10 à 14).

Pour respecter l'hypothèse liée à la forte répétition des comportements réguliers dans les traces d'exécution issues des cas d'utilisation de validation ou de tests d'endurance, la construction des jeux de données synthétiques est faite de manière à présenter un nombre important de fenêtres générées à partir des distributions étiquetées régulières et un faible nombre de fenêtres générées à partir des distributions irrégulières.

Pour générer les variantes de jeux de données synthétiques utilisées dans nos expériences, nous avons fait varier les valeurs des paramètres comme suit :

- le nombre de types d'événements : 10, 100, 1000. Nous avons choisi ces valeurs par analogie aux nombres de types d'événements contenus dans les jeux de données réels.
- le nombre de distributions d'événements : 10 au total, dont 7 sont considérées régulières, et 3 irrégulières.
- le nombre de fenêtres : 10000 dont 700 fenêtres sont construites à partir des distributions irrégulières.

Nous avons choisi de construire 10 distributions, dont 7 pour représenter les distributions régulières et 3 les distributions irrégulières, parce que ces valeurs sont proches de celles observées dans les données réelles.

Les jeux de données synthétiques ont été construits de manière à ce que la proportion des fenêtres étiquetées irrégulières soit plus importante que dans les données réelles. Ceci permet d'avoir des chiffres de détection plus précis et éviter les cas où on obtient $1/0$ comme rapport de détection. Par ailleurs, comme les fenêtres irrégulières ne sont ni connues à l'avance, ni apprises par l'algorithme de détection, les expériences sur les données synthétiques reflètent bien les conditions réelles de détection.

Les jeux de données synthétiques ont été générés et analysés sur un PC équipé d'un processeur Intel Xeon E5345 de 8 cœurs cadencé à 2.27 GHz, de 8 Go de RAM et d'un disque dur de 72 Go.

		Réalité			
		vrai	faux		
Prédiction	vrai	TP	FP	précision $\frac{TP}{TP+FP}$	taux fausses alarmes $\frac{FP}{TP+FP}$
	faux	FN	TN	taux fausses omissions $\frac{FN}{FN+TN}$	valeur de prédictions négatives $\frac{TN}{FN+TN}$
		rappel ou sensibilité $\frac{TP}{TP+FN}$	taux faux négatifs $(1 - spécificité)$ $\frac{FP}{FP+TN}$		
		taux faux positifs $\frac{FP}{FP+TN}$	Taux vrai négatifs ou spécificité $\frac{TN}{FP+TN}$		

TABLE 5.1 – Matrice de confusion

Mesures d'évaluation

Pour évaluer la qualité de détection de notre approche sur les données synthétiques, nous avons utilisé des mesures et outils bien connus dans les domaines de l'apprentissage supervisé et de la classification : le *rappel*, la *précision* et la courbe *sensibilité/spécificité* encore désignée par courbe *ROC*. Leur expression est résumée dans la Table 5.1 appelée *matrice de confusion*

Dans cette matrice de confusion, TP désigne le nombre vrais positifs, autrement dit le nombre de fenêtres irrégulières correctement identifiées par notre approche dans notre contexte. TN désigne le nombre de vrais négatifs ou encore le nombre de fenêtres régulières correctement identifiées. FP désigne le nombre de faux positifs, soit le nombre de fenêtres régulières identifiées comme irrégulières. FN désigne le nombre de faux négatifs, soit le nombre de fenêtres irrégulières identifiées comme régulières. La *précision* représente le ratio des fenêtres d'événements correctement reportées comme irrégulières parmi les fenêtres identifiées comme irrégulières. Une valeur importante de précision signifie qu'un nombre élevé de fenêtres identifiées comme irrégulières sont effectivement des fenêtres construites à partir des distributions irrégulières. Il est à noter que la valeur de la précision est significative en fonction de la prévalence, autrement dit du pourcentage de fenêtres irrégulières contenues dans le jeu de données. Ainsi une précision de 70% pour une prévalence de 10% est plus significative qu'une précision de 90% pour une prévalence de 1%.

Le *rappel* représente le ratio de fenêtres d'événements irrégulières correctement identifiées dans l'ensemble des fenêtres irrégulières contenues dans le jeu de données. Une valeur élevée de rappel signifie que notre approche réussit à identifier une grande partie des fenêtres irrégulières contenues dans le jeu de données.

La courbe *sensibilité/spécificité* (courbe *ROC*) permet d'évaluer la qualité de la détection pour des va-

leurs de sensibilité et spécificité obtenues à partir de différentes valeurs de paramètres. Elle compare la capacité à fournir un résultat positif lorsqu'une hypothèse est vérifiée (sensibilité) à l'opposé de la spécificité qui mesure la capacité à fournir un résultat négatif lorsqu'une hypothèse n'est pas vérifiée. En résumé, la courbe *ROC* est la courbe de la fonction :

$$sensibilité = f(1 - spécificité)$$

La courbe *ROC* se présente sous la forme d'une courbe où les valeurs en abscisse représentent les taux de faux positifs et les valeurs en ordonnée représentent le taux de vrais positifs. Une illustration de la courbe *ROC* est présentée à la Figure 5.2

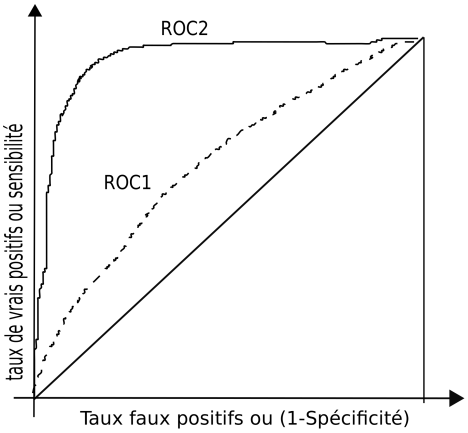


FIGURE 5.2 – Courbes *ROC* : Comparaison des performances de détecteurs. Plus une courbe est proche de la diagonale qui représente la détection aléatoire, moins le détecteur associé est de bonne qualité. Le détecteur associé à la courbe ROC2 est meilleur que celui associé à la courbe ROC1.

Les évaluations de qualité de détection de notre approche ont porté sur plusieurs critères parmi lesquels :

- la taille de l'espace d'apprentissage ;
- le nombre de types d'événements ;
- le bruit dans les données.

Pour chacun de ces critères d'évaluation, nous étudierons la qualité de détection de notre approche en fonction de deux paramètres importants de l'algorithme de détection *LOF* que sont :

- la taille du voisinage k à partir de laquelle les valeurs de densité sont calculées ;
- le seuil de détection ou score d'anomalie au delà duquel une fenêtre d'événements est considérée irrégulière.

Rappel du modèle de détection associé à notre approche

Notre approche de détection de fenêtres irrégulières consiste à attribuer à chaque fenêtre d'événements un score d'anomalie suivant le principe de l'algorithme de détection d'anomalie LOF. À cet effet, plus la distribution d'événements d'une fenêtre dévie des distributions associées aux régimes réguliers, plus le score d'anomalie attribué à cette fenêtre est élevé. Moins la distribution des événements d'une fenêtre dévie des distributions associées aux régimes réguliers, moins élevé est le score d'anomalie attribué à la fenêtre d'événements. Pour décider si une fenêtre est irrégulière (suspecte) ou pas, nous fixons une valeur de seuil

d'anomalie (ou seuil de LOF). Le choix de la valeur du seuil possède un impact important sur la qualité de détection. En général, plus le seuil est élevé, plus la précision est élevée, mais plus la sensibilité diminue. De même, moins le seuil est élevé, plus la sensibilité augmente mais par contre la précision diminue. Une illustration des valeurs de LOF associées aux fenêtres d'événements est présentée à la Figure 5.3. Cette figure présente les valeurs de LOF attribuées aux fenêtres d'événements indexées par leur timestamp de début sur l'axe des abscisses. Une fenêtre est considérée irrégulière si la valeur de LOF associée à son timestamp est au dessus du seuil d'anomalie. Si la valeur de LOF associée à une fenêtre est inférieure au seuil d'anomalie, alors la fenêtre est considérée comme régulière.

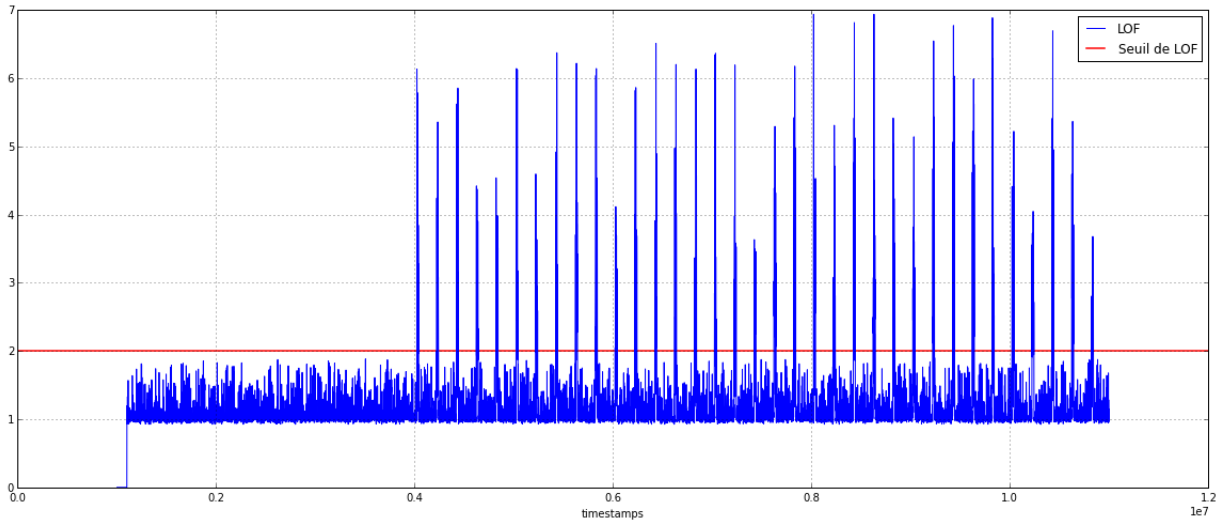


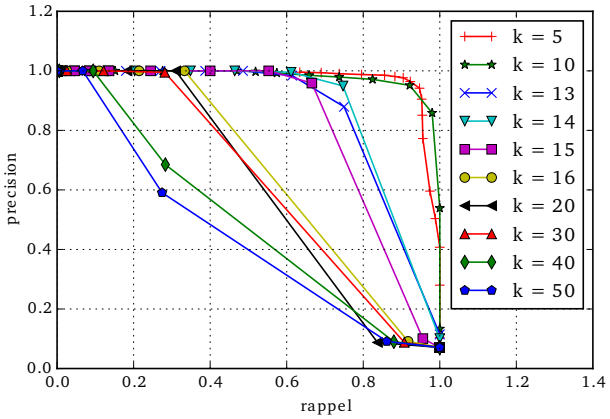
FIGURE 5.3 – Valeurs de LOF en fonction des fenêtres d'événements. Sur cette figure le seuil de LOF est fixé à 2. Les fenêtres dont le timestamp est associé à une valeur de LOF inférieure à 2 sont considérées comme correctes et les fenêtres pour lesquelles le timestamp est associé à une valeur de LOF supérieure à 2 représentent des anomalies.

5.1.2 Évaluation de la qualité de détection en fonction de la taille des données d'apprentissage

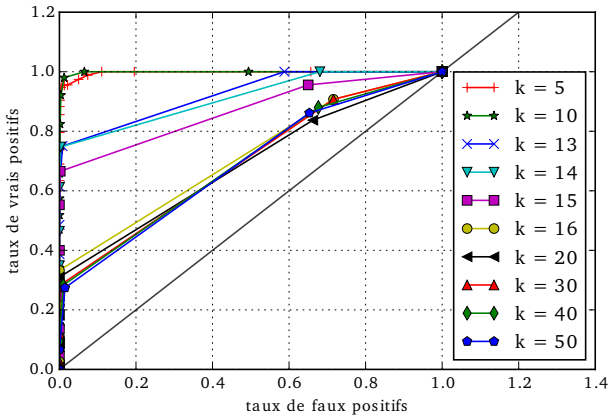
Cette expérience a porté sur l'évaluation de la qualité de détection en fonction de la période d'apprentissage. Pour ce faire, nous avons généré des jeux de données synthétiques sur 10 types d'événements, 10 distributions, 10000 fenêtres d'événements dont 700 fenêtres irrégulières. Pour cette expérience, nous n'avons pas introduit de bruit dans les jeux de données utilisés. Nous avons utilisé notre approche pour identifier les fenêtres irrégulières et par la suite, nous avons calculé le nombre de vrais positifs, de faux positifs, les valeurs de précision, de rappel en fonction de la taille de la période d'apprentissage. Les résultats obtenus sont visualisés par les courbes de précision/rappel suivant différentes tailles de l'ensemble d'apprentissage, différentes tailles de voisinage (k) et différents seuils de LOF.

Ces courbes sont présentées sur les Figures 5.4, 5.5.

Sur la figure 5.4, notre approche présente de bonnes performances de détection lorsque la taille de voisinage (k) est de 5. Cela s'observe par une précision qui chute à moins de 80% lorsqu'on atteint 85% de



(a) Courbe de précision en fonction du rappel pour un espace d'apprentissage de 100 fenêtres



(b) Courbe ROC pour un espace d'apprentissage de 100 fenêtres.

FIGURE 5.4 – Évaluation sur un espace d'apprentissage de 100 fenêtres en fonction de k

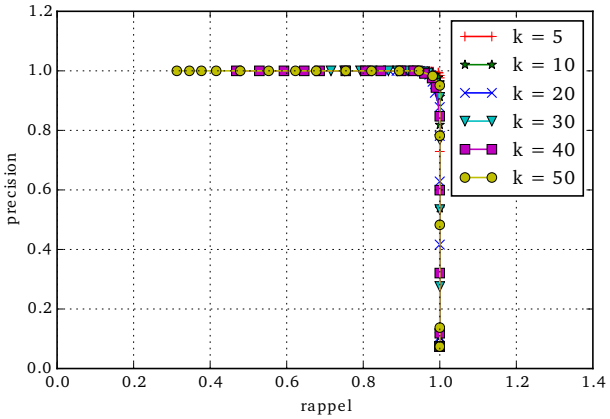
rappel sur la Figure 5.4a (soit un seuil de détection optimal de 2 pour une précision de 83% et un rappel de 83%) et un taux de vrais positifs de 1 pour un seuil de faux négatifs de moins de 20% sur la Figure 5.4b. Mais lorsque k prend une valeur supérieure à 15, notre approche présente des faibles performances de détection ainsi qu'on peut observer sur les courbes de précision/rappel et *ROC* pour $k > 15$. Ces résultats s'observent aussi à travers les valeurs de *AUC* (Area Under Curve) en fonction des valeurs de k présentées à la Table 5.2.

k	AUC
5	0.9969348602484472
10	0.9982273291925465
13	0.9249118788819876
14	0.9133378105590062
15	0.868776397515528
16	0.7149343021943069
20	0.6901214285714286
30	0.6962489130434784
40	0.6935717391304348
50	0.6880509316770186

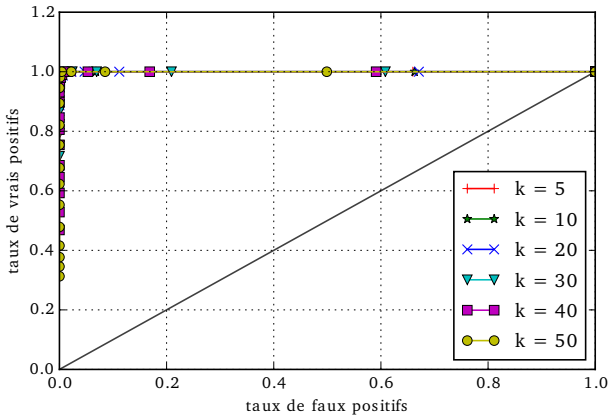
TABLE 5.2 – Valeurs AUC en fonction de la taille du voisinage k

Avant d'expliquer ces résultats, nous présentons les résultats obtenus pour un espace d'apprentissage constitué de 500 fenêtres.

Sur la Figure 5.5, nous avons représenté les performances obtenues sur le même jeu de données dans le cas où la période d'apprentissage est constituée de 500 fenêtres d'événements. On observe que notre approche présente de bonnes performances de détection quelle que soit la valeur de k sélectionnée. Les



(a) Courbe de précision en fonction du rappel pour un espace d'apprentissage de 500 fenêtres



(b) Courbe ROC pour un espace d'apprentissage de 500 fenêtres.

FIGURE 5.5 – Évaluation sur un espace d'apprentissage de 500 fenêtres en fonction de k

courbes de précision/rappel illustrées sur la Figure 5.5a montrent des valeurs de précision de 1 pour des valeurs de rappels quasiment égales à 1. Les courbes ROC de la Figure 5.5b présentent un taux de vrais positifs de 1 pour moins de 5% de faux positifs. Ces résultats montrent que sur un espace d'apprentissage constitué de 500 fenêtres sur ce jeu de données, notre approche détecte de manière quasi parfaite des fenêtres associées aux distributions irrégulières (soit en moyenne une précision de 98.2% pour un rappel de 98.1%).

La compréhension de ces résultats repose principalement sur la taille et la structure de l'espace d'apprentissage. Dans ces expériences, l'espace d'apprentissage est constitué du même nombre de fenêtres associées à chacune des (7) distributions régulières. Dans nos données synthétiques, les fenêtres qui constituent l'espace d'apprentissage sont uniformément associées aux différentes distributions régulières. Dans un espace d'apprentissage de 100 fenêtres, environ 14 fenêtres sont associées à chacune des 7 distributions régulières. Pour une taille de voisinage k de 5 à 14, les fenêtres sélectionnées dans le voisinage partagent la même similarité et de ce fait forment des clusters corrects pour la détection des anomalies. Pour des valeurs de $k > 14$, les fenêtres sélectionnées pour constituer les clusters associés aux distributions régulières commencent à être polluées par les fenêtres issues d'autres distributions et donc ne partageant toute la même similarité. Ce fonctionnement est illustré à la Figure 5.6.

Sur ce jeu de données, pour les valeurs de $k > 14$, l'espace d'apprentissage n'est plus suffisant pour distinguer les fenêtres régulières des fenêtres irrégulières, cela se traduit par des valeurs de LOF similaires entre les fenêtres régulières et les fenêtres irrégulières (illustration Figure 5.7). De façon générale, pour obtenir de bonne performance de détection, il faudrait que, pour tout régime régulier r_i, le nombre de fenêtres d'apprentissage $|F_{r_i}|$ relatives à un régime r_i soit supérieur à k.

$$|F_{r_i}| \geq k$$

Par contre pour une taille d'apprentissage de 500 fenêtres d'événements, les valeurs de k prises entre 5 et 50 ne présentent aucune difficulté pour construire des voisinages de fenêtres qui partagent la même similarité, ce qui rend stable les performances de détection de notre approche quelle que soit la valeur de k comprise entre 5 et 50.

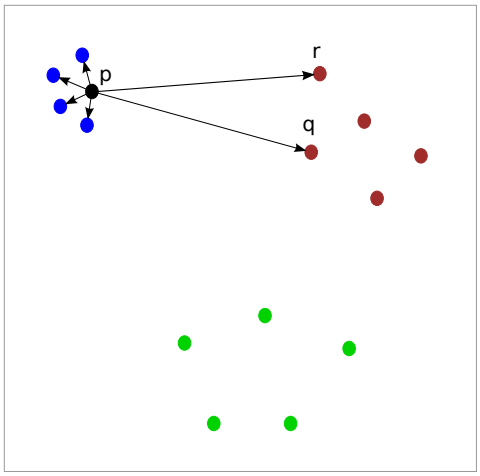


FIGURE 5.6 – Sur cette figure, un voisinage de taille 4 ($k = 4$) ne sélectionne que les points proches de p, alors qu'un voisinage de taille 5 pollue le voisinage proche de p en incluant le point q qui est très distant de p et donc pas similaire à p, ce phénomène s'accentue lorsque la taille du voisinage augmente

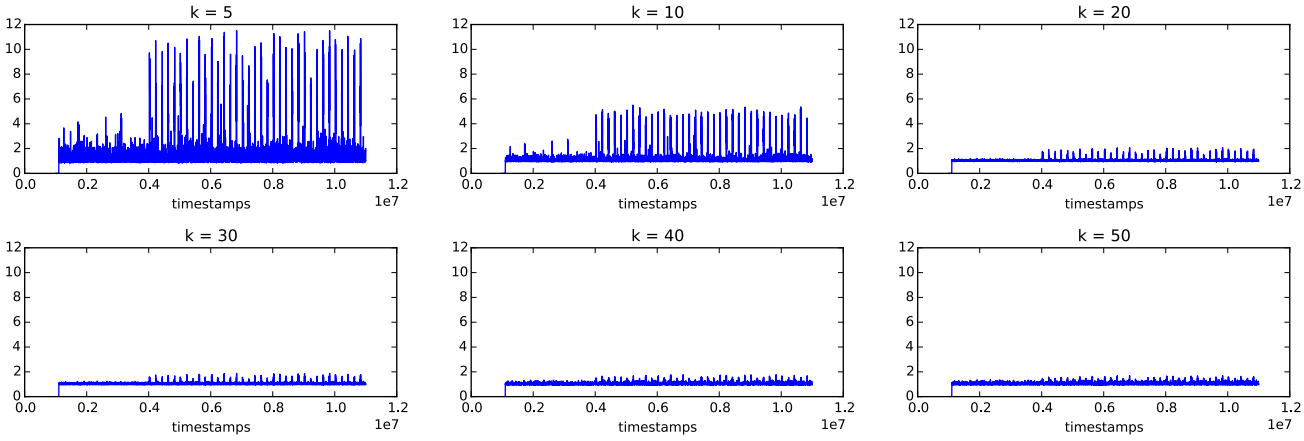
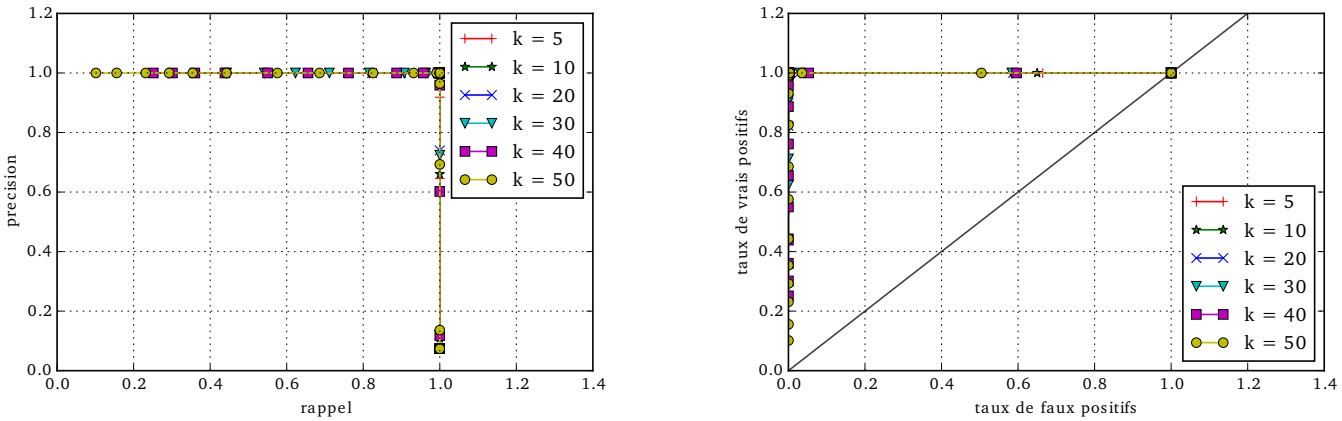


FIGURE 5.7 – Scores d'anomalies attribués aux fenêtres pour une taille d'apprentissage de 100 fenêtres

5.1.3 Évaluation de la qualité de détection en fonction du nombre de types d'événements

Dans cette sous-section, nous présentons les performances de détection de notre approche en faisant varier le nombre de types d'événements considérés sur les jeux de données synthétiques. Pour cette évaluation, nous avons généré des jeux de données synthétiques à partir de 1000 types d'événements. Nous avons choisi 1000 comme nombre de types d'événements par analogie au nombre moyen de types d'événements observés suivant une analyse fine de certaines traces d'exécution. Dans les jeux de données synthétiques générés, nous avons maintenu un espace d'apprentissage de 500 fenêtres d'événements, et nous n'avons pas inséré de bruit dans les données lors de la génération.

Une observation de notre algorithme de génération de données synthétiques montre que les données générées sur un espace de 1000 d'événements sont organisées en clusters bien séparés. Ces clusters sont construits de telle sorte qu'ils partagent un faible nombre de types d'événements. Cette séparation claire entre les clusters confère à notre approche de réduction de traces d'exécution de bonnes performances, en termes de précision et de rappel, pour la détection des fenêtres d'événements associées à des distributions irrégulières comme le montre la Figure 5.8.



(a) Courbe de précision en fonction du rappel

(b) Courbe ROC

FIGURE 5.8 – Évaluation sur des jeux de données construits à partir de 1000 types d'événements

Si une séparation nette entre les distributions d'événements (clusters) en fonction des types d'événements dans les jeux de données synthétiques est observée, ce n'est pas toujours le cas pour les jeux de données réels. Dans les traces d'exécution, certains types d'événements présentent un même suffixe et ne diffèrent que par la valeur attribuée à un paramètres qui leur est associé.

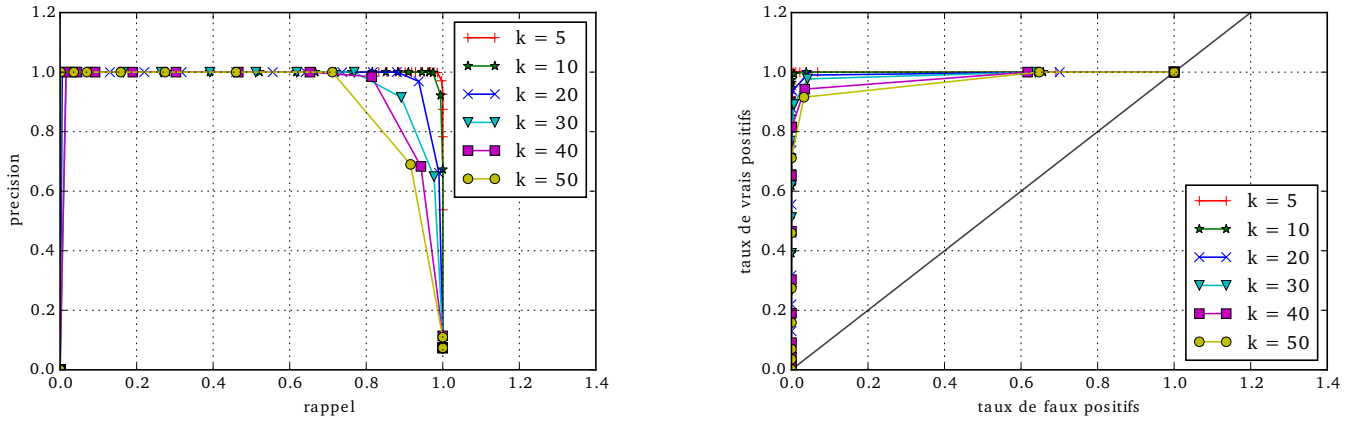
Exemple 7. *Trois instances d'événements de changement de contexte pour lequel les paramètres sont différents :*
__switch_to : SE-Audio-DtoM
__switch_to : SE-Audio-Mixer
__switch_to : SE-Audio-Mixer2

Une analyse fine de la trace d'exécution consiste à considérer toute différence pour distinguer les types

d'événements donnant lieu à des distributions d'événements spécifiques et par conséquent difficile à regrouper pour identifier des régimes d'exécution. Une agrégation de certains types d'événements (par exemple à partir du préfixe) permettrait d'exprimer des distributions générales favorisant l'identification des régimes d'exécution à travers leur regroupement.

Nous nous sommes intéressés aux performances de détection de notre approche dans un contexte d'analyse requérant l'agrégation des types d'événements. Nous avons réalisé des expériences sur les jeux de données synthétiques générés à partir de 1000 types d'événements pour lesquels nous avons effectué une agrégation des 1000 types d'événements pour obtenir un petit nombre d'événements.

Les résultats des fenêtres d'événements associées à des distributions étiquetées irrégulières présentées à la Figure 5.9 montrent que pour une importante agrégation conduisant 5 types d'événements, notre approche présente de manière générale de bonnes performances de précision et de rappel pour lesquelles le paramètre k présente une faible influence.



(a) Courbe de précision en fonction du rappel

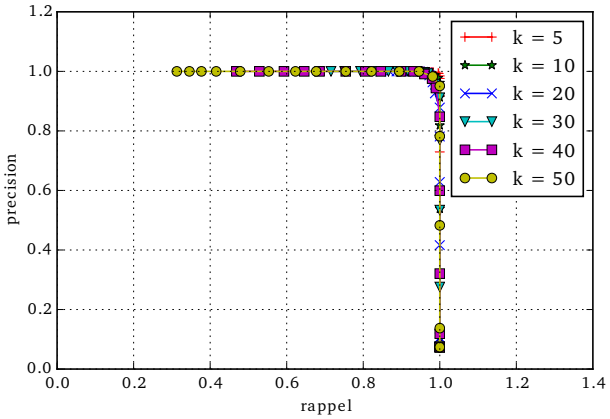
(b) Courbe ROC

FIGURE 5.9 – Évaluation sur des jeux de données construits à partir de 5 type d'événements agrégés.

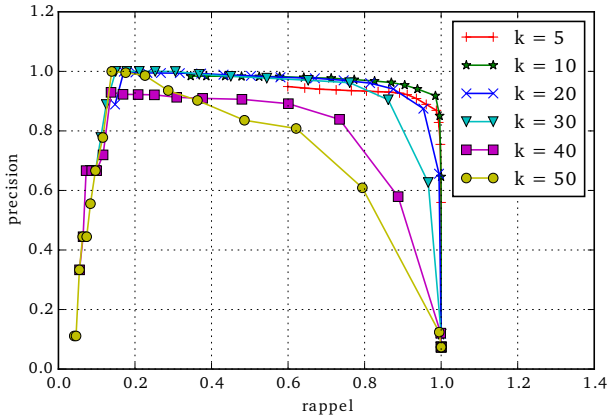
5.1.4 Évaluation de la qualité de détection en fonction du bruit dans les fenêtres d'événements

Les expérimentations présentées dans cette sous-section ont pour objectif d'évaluer la qualité de détection de notre approche en fonction du bruit dans les données. Pour réaliser ces expérimentations nous avons introduit du bruit dans les fenêtres d'événements. Le bruit consiste à modifier la distribution des événements dans les différentes fenêtres tout en s'assurant que la divergence entre les distributions bruitées et la distribution de base ne dépasse pas un certain seuil. Nous avons fait varier le seuil de divergence généré par le bruit de 0.05 à 0.5 dans les jeux de données sur lesquelles nous avons évalué la qualité de détection de notre approche. Les jeux de données utilisés pour cette expérience ont été générés sur 10 types d'événements. La taille d'apprentissage choisie est de 500 fenêtres d'événements.

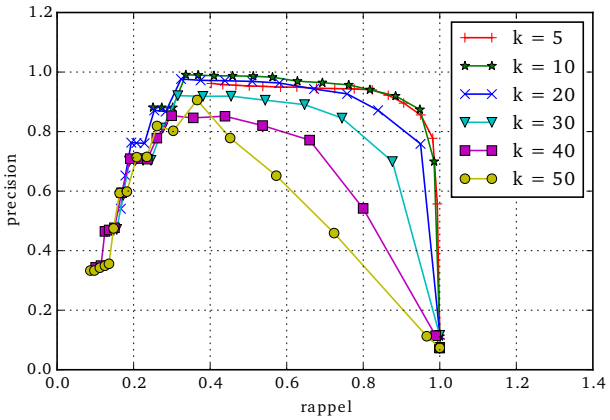
Suivant la Figure 5.10, on observe que la qualité de la détection des fenêtres irrégulières décroît suivant le pourcentage de bruit dans les données. Pour un seuil de bruit de 0.05 dans les données, la qualité de



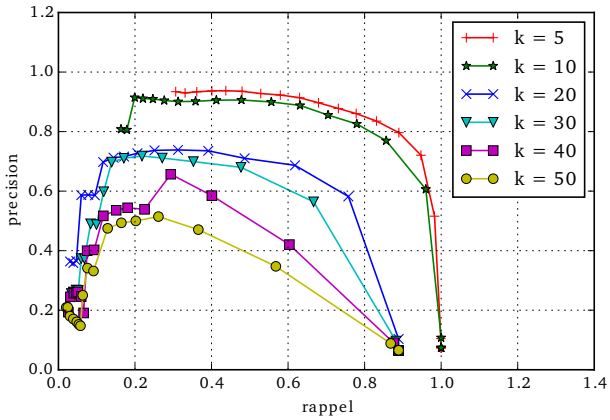
(a) Courbe de précision en fonction du rappel pour un seuil de bruit de 0.05 dans les données.



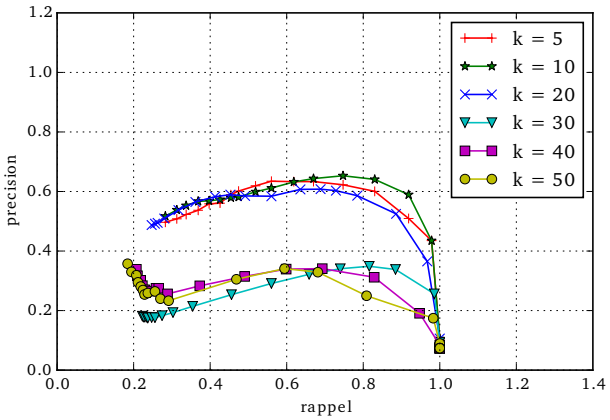
(b) Courbe de précision en fonction du rappel pour un seuil de bruit de 0.1 dans les données.



(c) Courbe de précision en fonction du rappel pour un seuil de bruit de 0.2 dans les données.



(d) Courbe de précision en fonction du rappel pour un seuil de bruit de 0.3 dans les données.



(e) Courbe de précision en fonction du rappel pour un seuil de bruit de 0.5 dans les données.

FIGURE 5.10 – Courbe de précision en fonction du rappel en fonction du bruit dans les données

détection reste correcte pour les valeurs k considérées. On obtient une précision comprise entre 95% et 86% pour un rappel compris entre 98% et 86% en fonction de k. On observe aussi sur les Figures 5.10b, 5.10c, 5.10d et 5.10e qu'en présence du bruit dans les données, la qualité de la détection devient sensible aux valeurs de k quand bien même le taille de la période d'apprentissage est importante. En effet, les courbes de précision/rappel présentent une dégradation importante de la précision et du rappel pour de grandes valeurs de k (30 à 50). Ce comportement s'explique par le fait que dans des données comportant du bruit, plus on sélectionne des points pour identifier les clusters associés aux distributions régulières, plus on introduit du bruit dans ces clusters, ce qui réduit la qualité des clusters identifiant les distributions régulières et donc la capacité de détection des distributions irrégulières.

5.2 Expérimentations sur les jeux de données réels

Dans cette section, nous présentons les expériences réalisées sur les données réelles qui sont constituées de traces d'exécution (applicatives) propres au framework GStreamer et de traces d'exécution provenant d'un système embarqué. Ces expériences ont pour objectif de valider notre approche de réduction sur des cas d'utilisation réels. Nous présentons la qualité de détection de notre approche de réduction sur les traces réelles ainsi que le taux de réduction obtenu sur les jeux de données réelles.

5.2.1 Expérimentations sur les traces applicatives

Les traces applicatives sur lesquelles nous avons réalisé nos expériences sont des traces générées par l'application *gst-launch*. L'application *gst-launch* est un outil du framework multimédia GStreamer qui permet de construire et d'exécuter un pipeline de composants pour le décodage audio/vidéo. Nous rappelons que GStreamer est un framework multimédia open-source dédié à l'implémentation des applications de traitement de flux multimédia. Nous qualifions les traces d'exécution produites par l'application *gst-launch* de traces applicatives parce que ces traces d'exécutions sont constituées uniquement des événements générés par les composants de l'application. Pour nos expériences, nous avons généré des traces d'exécution de cette application au cours du processus de décodage audio/vidéo des fichiers de 10 minutes, de 6 heures et de 27 heures de vidéo. Pendant le décodage audio/vidéo de ces fichiers, nous avons lancé à certaines périodes précises une autre application gourmande en ressource processeur pour perturber le processus de décodage audio/vidéo.

Présentation de la preuve de concept

La première expérience a été réalisée sur une vidéo de 10 minutes pour laquelle, au cours du décodage audio/vidéo, nous avons introduit des perturbations de 10 secondes toutes les 2 minutes. Cette expérience a consisté à réaliser une preuve de concept dans le but d'observer le comportement de notre approche de détection des irréguliers en fonction du décodage audio/vidéo en période de perturbations ou non. Pour cela, nous avons réalisé une analyse de la trace générée en considérant comme période d'apprentissage les 30 premières secondes de la trace. Le comportement de notre approche se caractérise par les différentes

valeurs de LOF attribuées aux différentes fenêtres dans la trace d'exécution. Les résultats de notre analyse du comportement des valeurs de LOF en fonction des périodes de décodage sont présentés à la Figure 5.11.

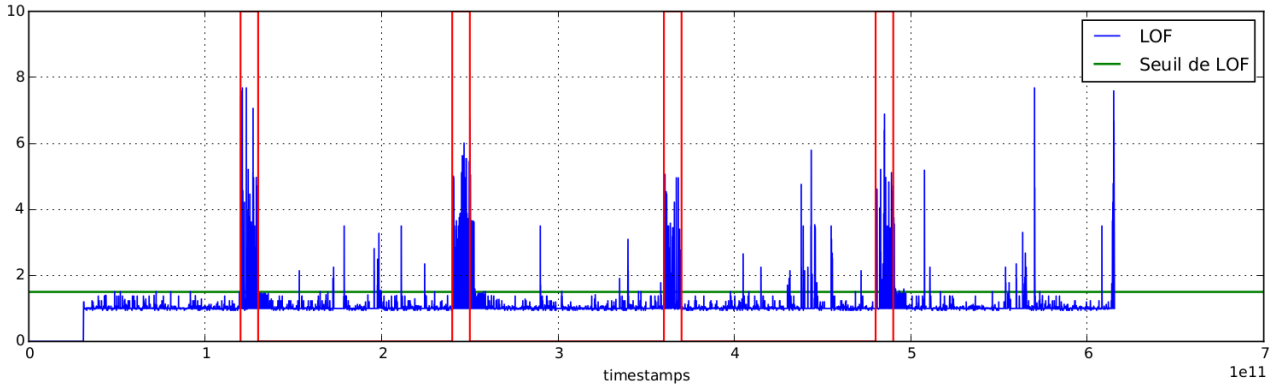


FIGURE 5.11 – Courbe des scores d'anomalies des fenêtres d'événements d'une trace d'exécution

La Figure 5.11 présente les valeurs de LOF en fonction des timestamps de début des fenêtres d'événements dans la trace d'exécution. Les lignes rouges délimitent les périodes d'exécution au cours desquelles les perturbations ont été effectuées. Les pics des valeurs de LOF indiquent les fenêtres d'événements associées à des régimes de fonctionnement irréguliers par rapport aux régimes réguliers identifiés pendant la période d'apprentissage. On observe sur cette figure que la courbe des valeurs de LOF présente des zones de pics de valeurs qui correspondent aux zones de perturbations. En effet, au cours de ces périodes de perturbations, le processus présente des difficultés à réaliser correctement le décodage audio/vidéo. Cela se traduit par des micro-interruptions au niveau des rendus vidéo et sonore. Par contre, pendant les périodes de fonctionnement normal de l'application, les valeurs de LOF sont en dessous d'un certain seuil (représenté par la ligne verte sur la Figure 5.11). De cette expérience, il se dégage que notre approche permet d'identifier automatiquement et sans connaissance a priori, les régimes de fonctionnement correspondant aux comportements anormaux ou irréguliers d'une application.

Évaluation de la qualité de détection

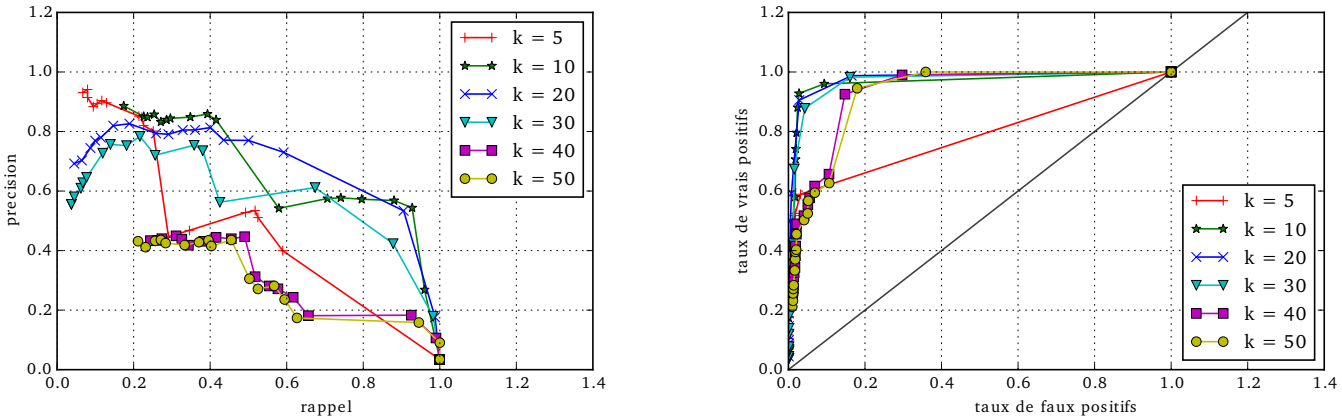
Nous avons mené une expérience sur la trace applicative de 10 minutes générée à partir de *gst-launch* sujet aux perturbations contrôlées. Le but de cette expérience a consisté à évaluer la qualité de détection des comportements anormaux correspondant aux périodes de perturbation provoquées pendant le décodage. L'évaluation de la qualité de détection a été réalisée en fonction de plusieurs paramètres parmi lesquelles :

- la taille de voisinage,
- le seuil de valeurs de LOF permettant d'identifier les régimes réguliers,
- la taille de la période d'apprentissage,
- la taille des fenêtres d'événements.

Évaluation de la qualité de détection en fonction de la période d'apprentissage

L'évaluation de la qualité de détection de notre approche en fonction de l'espace d'apprentissage a été réalisée en faisant varier la taille de la période d'apprentissage sur les données. Pour ce faire, nous avons utilisé la trace d'exécution issue du décodage vidéo de 10 minutes. Nous avons considéré comme

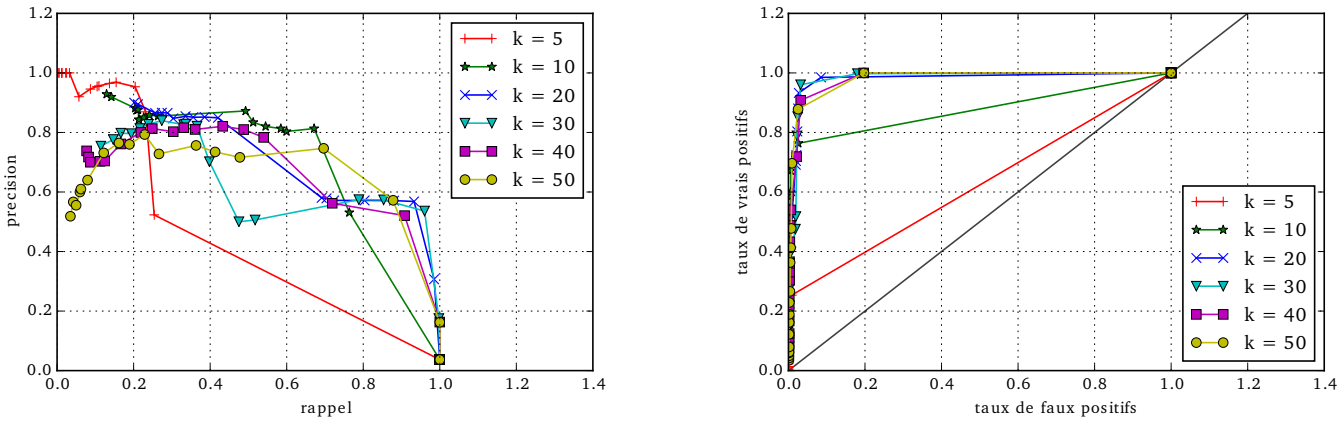
données d'apprentissage, les parties de la trace situées dans les 30, 60, 90 et 120 premières secondes de données, périodes au cours desquelles aucune perturbation contrôlée n'a été effectuée. Nous avons fixé la taille des fenêtres à 40 millisecondes, et suivant les différentes périodes d'apprentissage, nous avons évalué la précision et le rappel pour différents seuils de LOF



(a) Courbes de précision en fonction du rappel

(b) Courbes ROC

FIGURE 5.12 – Qualité de détection pour une période d'apprentissage de 30s



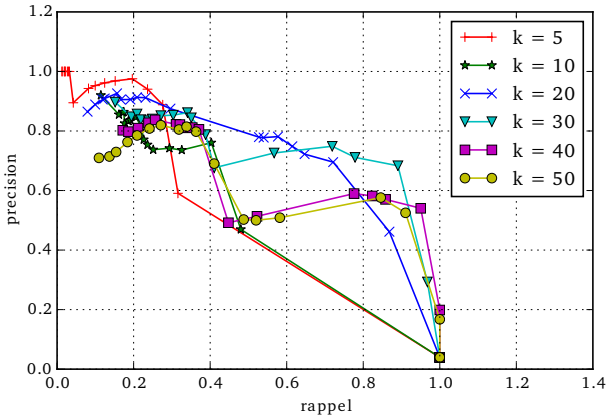
(a) Courbes de précision en fonction du rappel

(b) Courbes ROC

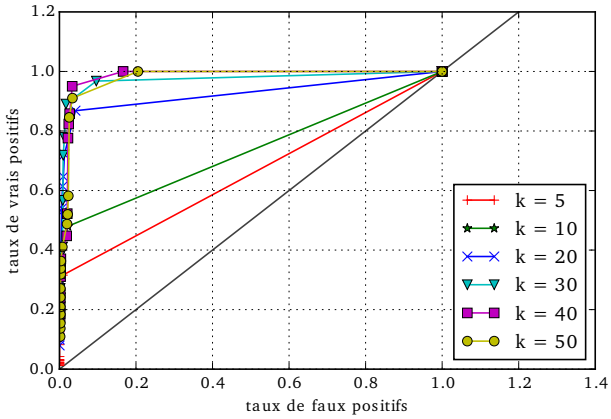
FIGURE 5.13 – Qualité de détection pour une période d'apprentissage de 60 secondes

Les Figures 5.12, 5.13, 5.14, 5.15 présentent respectivement les performances de détection (courbes précision/rappel et ROC) en fonction de différentes périodes d'apprentissage. D'après ces figures, on observe que les performances de détection augmentent en fonction de la durée (taille) de la période d'apprentissage pour certaines tailles de voisinage (valeurs du paramètre k), notamment 20, 30, 40 et 50. Par exemple pour le cas $k = 20$, les valeurs optimales de précision et rappel obtenues, en fonction de la taille de la période d'apprentissage, sont présentées dans la Table 5.3

Nous remarquons aussi que les courbes précision/rappel obtenues présentent une forme en escalier. Cette forme en escalier s'explique par la présence de pics de valeurs de LOF dans les périodes de fonction-

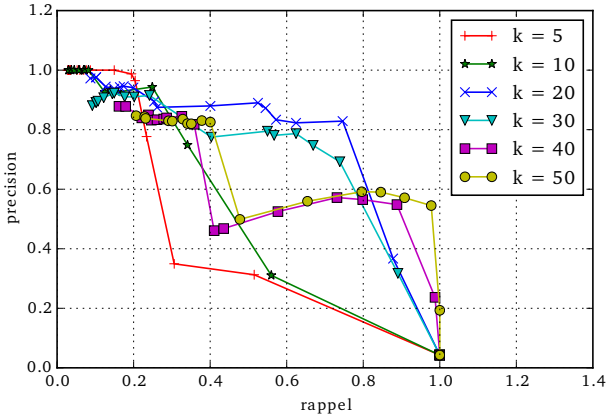


(a) Courbes de précision en fonction du rappel

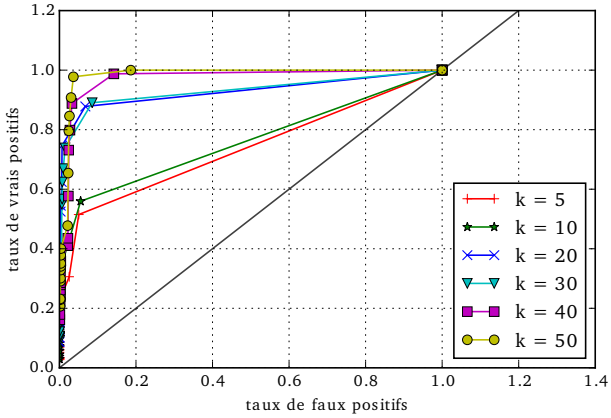


(b) Courbes ROC

FIGURE 5.14 – Qualité de détection pour une période d'apprentissage de 90 secondes



(a) Courbes de précision en fonction du rappel



(b) Courbes ROC

FIGURE 5.15 – Qualité de détection pour une période d'apprentissage de 120 secondes

Période d'apprentissage	Précision	Rappel
30 s	0.77	0.59
60 s	0.57	0.69
90 s	0.7	0.69
120 s	0.83	0.72

TABLE 5.3 – Valeurs de précision et de rappel pour $k = 20$ et en fonction de la taille de la période d'apprentissage

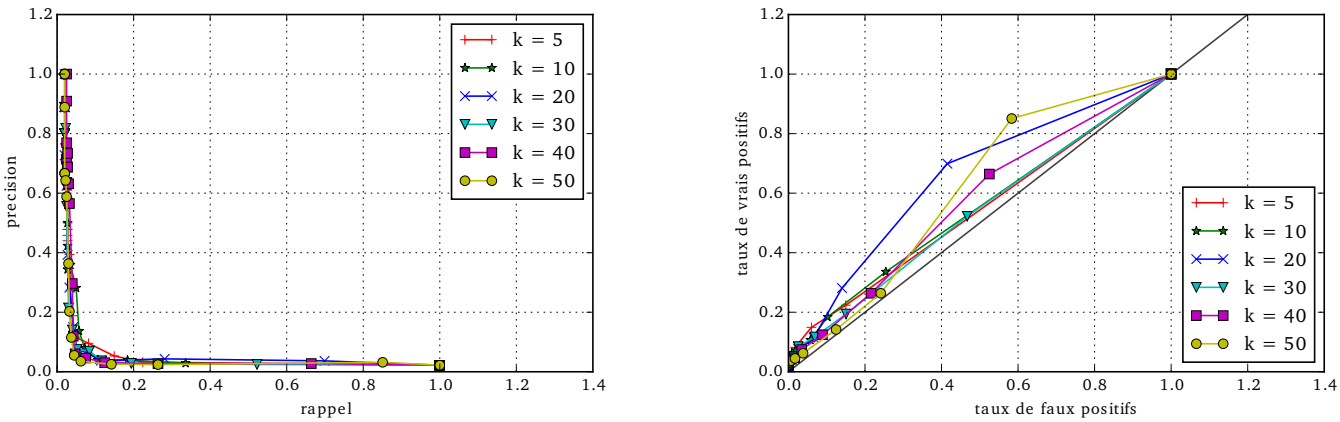
nement non perturbés. Ces pics de valeurs de LOF hors des périodes de perturbations constituent des faux positifs.

Une analyse fine des fenêtres d'événements détectées comme faux positifs montre que les distributions d'événements associées à ces fenêtres sont peu présentes dans la trace d'exécution. De même, ces distributions apparaissent très peu pendant la période d'apprentissage. Elles forment des petits clusters dont la taille en nombre de points est insuffisante pour constituer des régimes réguliers. Une solution consiste à utiliser une période d'apprentissage plus importante ou un apprentissage sur une trace d'exécution similaire mais sans aucune perturbation effectuée. Bien que la solution basée sur l'utilisation d'une période d'apprentissage plus importante réduirait le nombre de faux positifs, elle induirait une complexité importante dans le calcul des valeurs de LOF pour la détection des régimes irréguliers car la complexité du calcul des valeurs de LOF est quadratique en fonction de la taille de l'espace d'apprentissage.

Évaluation de la qualité de détection en fonction de la taille des fenêtres d'événements

Nous nous sommes ensuite intéressés à l'impact de la taille des fenêtres d'événements sur la qualité de détection de notre approche. Pour cela, nous avons fixé la période d'apprentissage à 120 secondes sur la base des résultats précédents, et nous avons fait varier la taille des fenêtres d'événements avec des valeurs prises entre 20, 40, 60, 80 millisecondes.

Les Figures 5.16, 5.17, 5.18, 5.19 présentent les courbes précision/rappel et ROC pour différentes tailles de fenêtres d'événements (resp. 20, 40, 60 et 80 ms) et suivant différentes valeurs du paramètres de voisinage k.

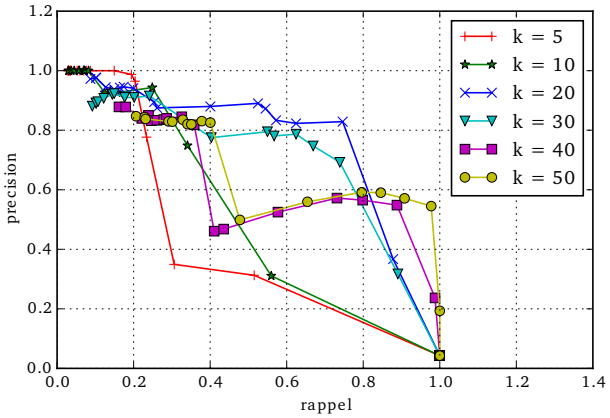


(a) Courbes de précision en fonction du rappel

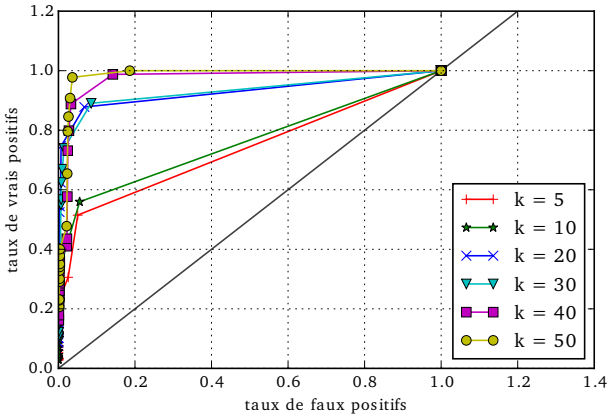
(b) Courbes ROC

FIGURE 5.16 – Qualité de détection avec une taille de fenêtres de 20ms

Les résultats obtenus montrent que notre approche est sensible à la taille des fenêtres d'événements. De manière générale, nous observons que notre approche de détection présente des bonnes performances de détection des régimes irréguliers lorsque les fenêtres d'événements ont une taille fixée à 40 ms (Figure 5.17), et des mauvaises performances de détection pour les tailles de fenêtres de taille différente de 40 ms. Ce comportement trouve des explications dans les propriétés de l'application tracée. L'application tracée dans ces expériences est une application de décodage audio-vidéo et pour lesquelles les contraintes

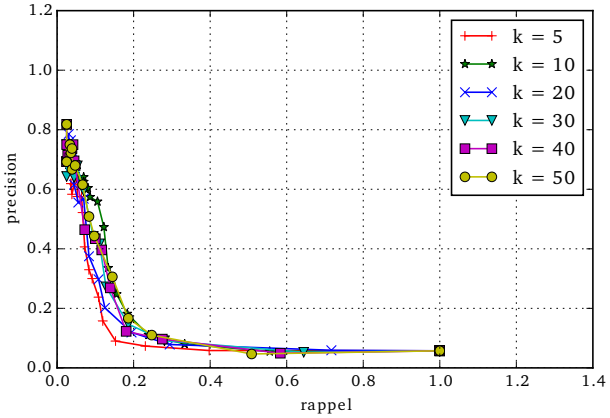


(a) Courbes de précision en fonction du rappel

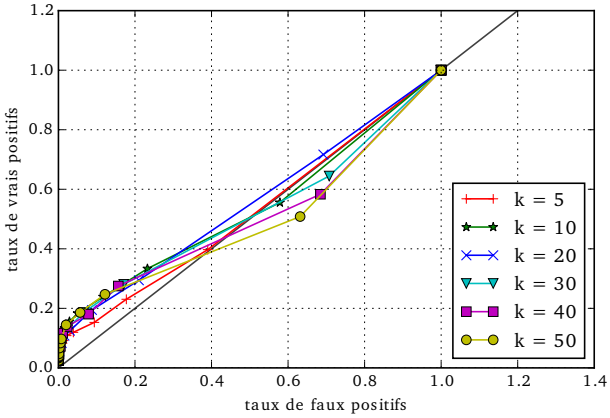


(b) Courbes ROC

FIGURE 5.17 – Qualité de détection avec une taille de fenêtres de 40ms

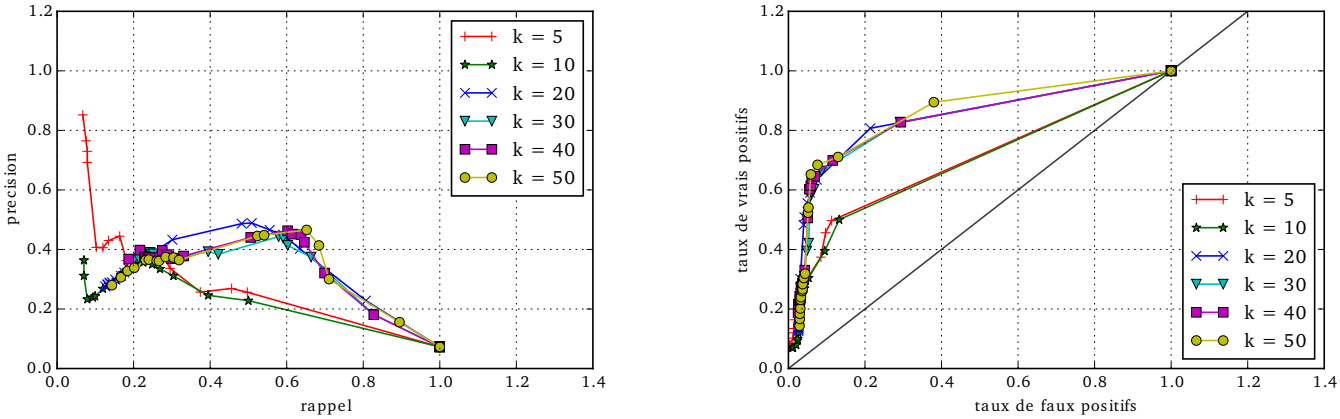


(a) Courbes de précision en fonction du rappel



(b) Courbes ROC

FIGURE 5.18 – Qualité de détection avec une taille de fenêtres de 60ms



(a) Courbes de précision en fonction du rappel

(b) Courbes ROC

FIGURE 5.19 – Qualité de détection avec une taille de fenêtres de 80ms

de fonctionnement sont telles que le décodage complet doit être réalisé sur un délai de 40 ms. À cet effet, le fait d'analyser les fenêtres d'événements de 40 ms conduit à de fortes chances de capturer les régimes effectifs de fonctionnement de l'application, notamment ceux liés au décodage complet des frames.

Ainsi, l'analyse sur des fenêtres de tailles inférieures à 40 ms consisterait à étudier les distributions d'événements associées à des parties de régimes qui composent les régimes de fonctionnement de l'application. L'analyse des valeurs de LOF obtenues montre que la majorité des fenêtres d'événements associées à ces pics de valeurs de LOF ne sont pas correctement alignées avec les instants de perturbation de l'application.

Évaluation de l'approche de détection sur une trace de taille importante

Pour évaluer la pertinence de notre approche de réduction sur des traces d'exécution générées sur des périodes d'exécution importantes (cas d'utilisation similaires aux tests d'endurances), nous avons analysé les performances de notre approche sur une trace d'exécution générée pour un décodage audio/vidéo de 6 heures. Au cours de ce processus de décodage, nous avons introduit des perturbations contrôlées suivant un schéma similaire à celui de l'expérience réalisée sur le décodage de la vidéo de 10 minutes. Ainsi après 5 minutes de décodage, nous avons introduit des perturbations d'une durée de 20 secondes toutes les 3 minutes. Le décodage audio/vidéo de 6h a produit une trace d'exécution d'une taille d'environ 5 Go. Pour analyser cette trace d'exécution, nous avons considéré uniquement les paramètres de notre approche pour lesquels nous avons obtenu de bonnes performances dans les expériences précédentes sur les données réelles. Nous avons fixé la taille des fenêtres d'événements à 40 ms, une valeur de k de 20 (taille du voisinage) et une période d'apprentissage de 5 minutes. Les performances de détection de notre approche sont présentées aux Figures 5.20, 5.21 et 5.22.

Les Figures 5.20 et 5.21 présentent les valeurs de précision et de rappel obtenues en fonction du seuil de LOF pour la détection des régimes irréguliers. On observe que pour un seuil de détection de 1.25, notre approche présente une précision de $82,15\%$ et un rappel de 98.37%. Ce qui signifie qu'environ 82% des fenêtres d'événements détectées comme irrégulières par notre approche sont effectivement des fenêtres

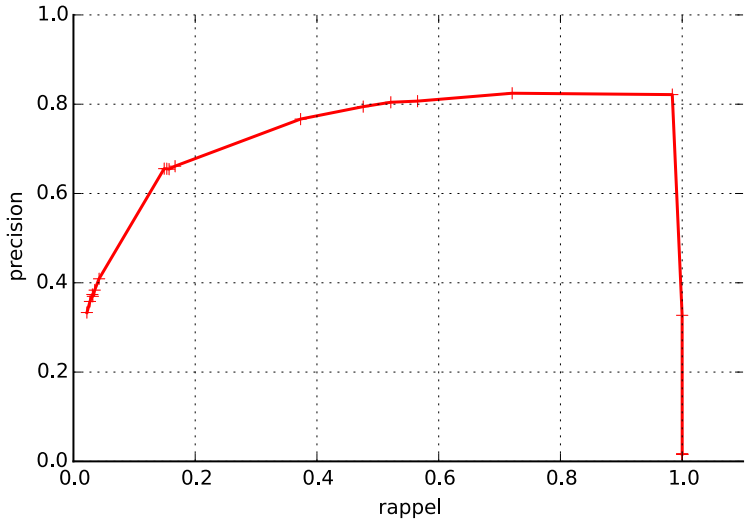


FIGURE 5.20 – Évaluation de la précision en fonction du rappel

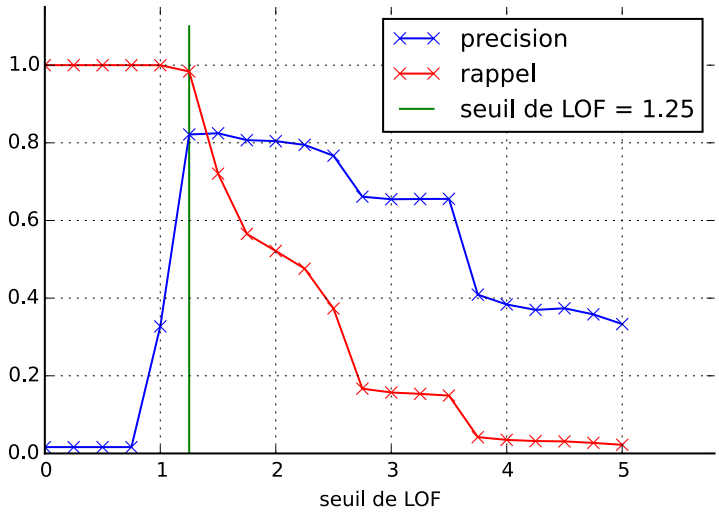


FIGURE 5.21 – Évaluation de la précision et du rappel

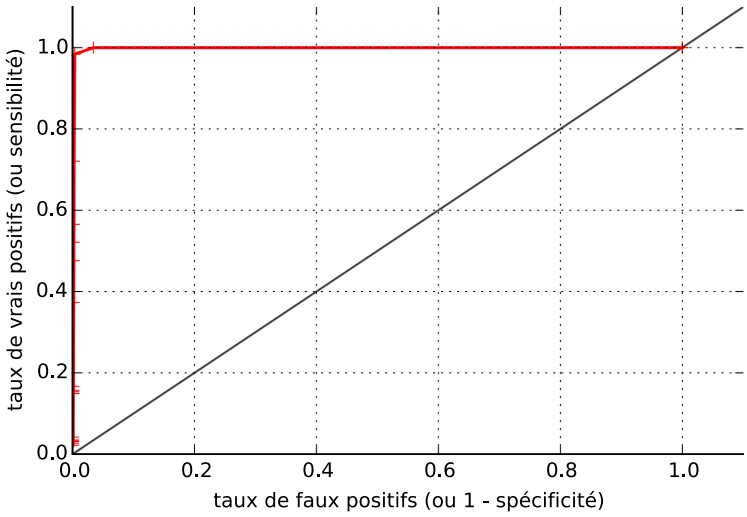


FIGURE 5.22 – Courbe ROC

d'événements sujettes aux perturbations effectuées, et $98,37\%$, soit la quasi-totalité des fenêtres d'événements sujettes aux perturbations ont été détectées.

La Figure 5.22 et la table 5.4 qui présentent respectivement la courbe ROC et les valeurs de sensibilité (vrais positifs) en fonction de l'anti-spécificité (1-spécificité ou taux de faux positifs) montrent que pour un seuil de détection supérieur à 1, notre approche présente un faible nombre de faux positifs, c'est à dire un faible nombre de fenêtres d'événements régulières mais détectées comme irrégulières.

LOF	$1 - Spécificité$	$Sensibilité$
0	1.0	1.0
0.25	1.0	1.0
0.5	1.0	1.0
0.75	1.0	1.0
1.0	0.0341	0.999
1.25	0.003	0.983
1.5	0.002	0.720
1.75	0.002	0.565
2.0	0.002	0.521
2.25	0.002	0.475
2.5	0.001	0.373
2.75	0.001	0.166
3.0	0.001	0.156
3.25	0.001	0.153
3.5	0.001	0.149
3.75	0.001	0.042
4.0	0.000	0.034
4.25	0.000	0.031
4.5	0.000	0.031
4.75	0.000	0.027
0.5	0.000	0.022

TABLE 5.4 – Valeurs de sensibilité et spécificité en fonction des seuils de LOF

Sur cette trace d'exécution, nous avons évalué le taux de réduction de notre approche, autrement dit le pourcentage de réduction de données que notre approche a permis d'obtenir. À cet effet, nous avons obtenu un taux de réduction de 97.8% en termes de taille de données et de nombre d'événements. Nous sommes donc passés d'une trace complète d'une taille de $4,77$ Go à une trace réduite de $104,45$ Mo contenant principalement les parties de la trace d'origine associées à des régimes irréguliers ou comportements anormaux. Sur cette trace d'exécution de 6 heures, notre approche de réduction permet d'obtenir une réduction de données d'un ordre de grandeur.

En général, il est préférable d'avoir une bonne précision qu'un bon rappel pour éviter les fausses alarmes dans les données détectées positifs. Dans notre cas, nous sommes intéressés aux paramètres de détection conduisant à un rappel exhaustif car nous souhaitons capturer toutes les occurrences des potentiels bogues dans la trace réduite et laisser le soin au développeur de décider de leur pertinence pour l'application.

L'approche de réduction proposée présente une bonne précision qui justifie la qualité de réduction de la trace d'exécution et un rappel plus important qui confirme la capture de la quasi-totalité des occurrences du bug dans la trace réduite.

Par ailleurs, nous avons évalué le débit de traitement de notre approche afin de le comparer au débit de production de trace de l'application. Notre approche de réduction, pour les paramètres d'analyses ci-dessus, présente un débit moyen de traitement de 3.65 Mbits/s contre un débit de génération de trace de 351.3 Kbits/s.

5.2.2 Expérimentations sur les traces des systèmes embarqués

Pour valider notre approche sur les traces générées par les systèmes embarqués, nous avons appliqué notre approche de réduction des traces d'application multimédia générées par la carte STiH416 développée pour STMicroelectronics. La carte STiH416 est le système embarqué qui équipe les Set-Top Box de 4^e génération. Elle est composée :

- d'un microprocesseur ARM Cortex A9 dual core,
- d'un GPU quad core pour les traitements 3D,
- d'un processeur ST40 pour le traitement de flux multimédia,
- des décodeurs vidéo et audio,
- des interfaces d'E/S (HDMI, audio, TV, USB, Ethernet, MMC/SDIO, SATA)

Les traces générées par les systèmes embarqués sont des traces constituées d'événements applicatifs (30%) et d'événements systèmes (70%). Les événements systèmes générés sont pour la plupart les changements de contexte, les interruptions, les appels de fonctions systèmes, etc.

Notre première expérience sur les traces d'exécution générées par la carte STiH416 a porté sur l'analyse d'une application d'enregistrement de flux audio-vidéo. Le cas d'utilisation utilisé pour cette expérience consistait à utiliser l'application *ts_record* pour enregistrer un flux audio/vidéo diffusé sur l'interface réseau de la carte vers un disque USB. La trace d'exécution a été générée pour 8 minutes d'exécution. Suite à l'application de notre approche d'analyse sur la trace générée, des pics de LOF ont été observés sur certaines fenêtres d'événements de la trace d'exécution. Une illustration des valeurs de LOF sur une partie de la trace d'exécution est présentée à la Figure 5.23.

Une analyse statistique (cf. Figure 5.24) des fenêtres d'événements associées aux pics de valeurs de LOF montre que les fenêtres pour lesquelles les valeurs de LOF sont supérieures à 2 présentent un nombre important de certains type d'événements qui apparaissent très peu ou pas du tout dans les fenêtres associées aux régimes réguliers. Sur la Figure 5.24, on observe que la fenêtre associée au pic de valeur LOF contient un nombre d'occurrences de *context-switch* (changement de contexte) élevé par rapport à la distribution des événements de *context-switch* contenues dans les fenêtres associées aux régimes réguliers.

Suite à cette observation, une analyse fine de la trace d'exécution montre que chacune des fenêtres détectées comme fenêtres de régimes irréguliers contient une occurrence de l'une des tâches *flush-8* et *flush-0*. L'exécution de ces tâches entraîne le transfert de données des buffers vers le disque USB. Ce transfert se traduit globalement dans la trace par une augmentation de l'activité de la tâche *usb_storage* et des interruptions sur l'interface USB (*ehci_hcd :usb3*) d'une part, et d'autre part par une diminution des

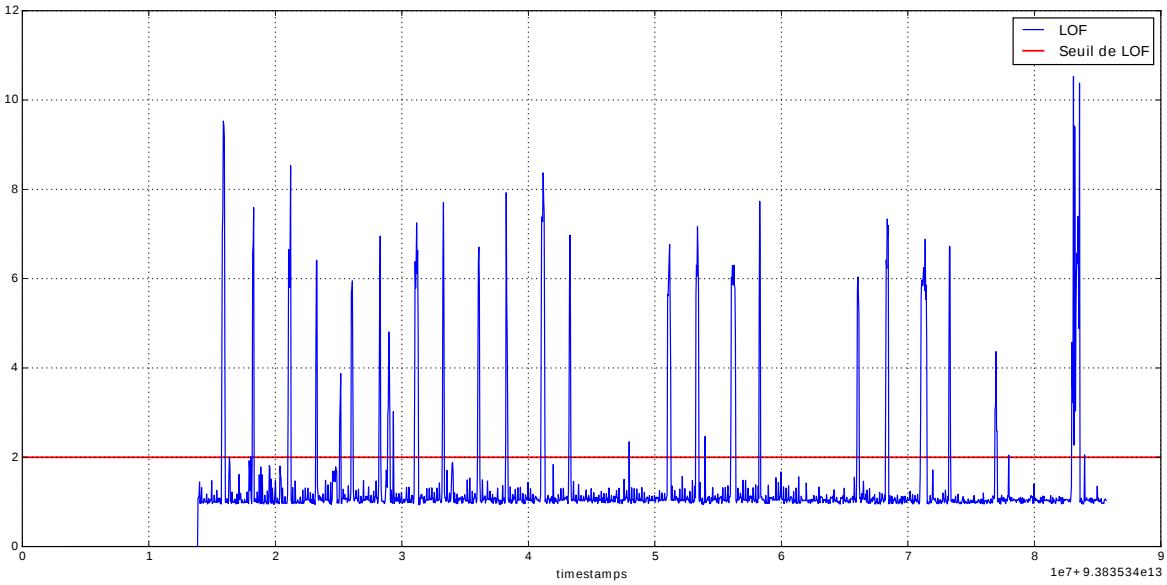


FIGURE 5.23 – Valeurs de LOF sur la trace générée par l'application *ts_record*

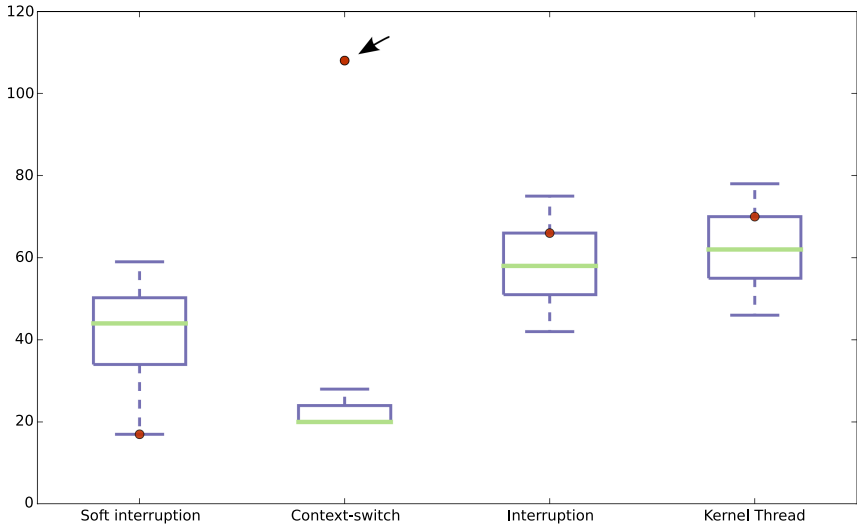


FIGURE 5.24 – Description des régimes irréguliers à l'aide des boîtes à moustaches : les boîtes à moustaches représentent la distribution des certains événements dans un régime régulier et les points représentent le nombre d'occurrences de ces événements dans le cas d'un régime irrégulier détecté.

interruptions sur l'interface réseau (*eth0*). Ces résultats d'analyse correspondent bien à un bogue identifié par les développeurs et prouve que notre approche peut retrouver effectivement les bogues des applications à partir de leur trace d'exécution.

Sur cette trace d'exécution, nous avons obtenu un taux de réduction de 88% avec un débit moyen de traitement de 21.8 Mbits/s pour un débit de génération de trace de 2 Mbits/s.

5.3 Conclusion sur les expériences et évaluations

Dans ce chapitre, nous avons montré de manière expérimentale que notre approche de réduction des traces d'exécution détecte réellement, à partir des traces d'exécution des applications, les régimes irréguliers de l'application ou les régimes d'exécution liés aux comportements anormaux de ces applications, et cela sans aucune connaissance a priori.

Les expériences réalisées sur les jeux données synthétiques et les jeux de données réels montrent que notre approche de réduction présente de bonnes performances de détection de régimes irréguliers lorsque la taille des données d'apprentissage est suffisamment grande. Ces performances de détection restent satisfaisantes suivant que l'on passe d'une analyse fine en termes de types d'événements dans les traces d'exécution à une analyse à gros grain. La qualité de détection de notre approche permet d'obtenir une réduction des données d'un ordre de grandeur. On obtient ainsi une trace d'exécution réduite constituée principalement des données pertinentes pour le débogage ou le profilage. De plus, notre approche fournit des intuitions sur le caractère anormal des fenêtres d'événements associées à des régimes irréguliers identifiés.

Par ailleurs ces expériences ont permis de montrer que l'approche de réduction que nous proposons présente un débit de traitement de données considérable par rapport au débit de génération des traces d'exécution des systèmes embarqués, lui permettant ainsi d'être réalisée à la volée.

CHAPITRE 6

Conclusion et perspectives

Le développement des fonctionnalités proposées par les systèmes embarqués a conduit à une importante complexification de leur architecture, rendant difficile les processus de débogage et profilage des applications. Les approches de débogage et profilage utilisées sur ces architectures reposent principalement sur la production et l'analyse des traces d'exécution. Cependant, dans certains cas d'évaluation des applications sur les systèmes embarqués, les traces d'exécution générées sont très importantes en termes de volume, au point de rendre leur exploitation difficile voire impossible.

Face aux difficultés liées aux volumes des traces d'exécution, nous proposons une approche de réduction en ligne des traces d'exécution. Cette approche de réduction repose sur l'observation que les applications, en particulier les applications multimédia, présentent beaucoup de régularité dans leur fonctionnement. Cette régularité se traduit par une redondance de l'information dans les traces d'exécution. Notre approche de réduction étudie cette redondance de l'information dans les traces d'exécution pour identifier les régimes réguliers des applications et retenir les parties de traces d'exécution qui présentent un intérêt pour le débogage ou le profilage.

Pour mieux cerner les motivations et le contexte de notre contribution, nous avons montré au Chapitre 2 que l'analyse des traces d'exécution se présente comme l'approche la mieux adaptée pour l'analyse des applications sur les systèmes embarqués. Cependant dans certaines conditions, les traces d'exécution produites peuvent devenir rapidement importantes en volume au point d'être inexploitable par les outils d'analyse des traces d'exécution. Une solution adaptée au problème de volumétrie des traces d'exécution est la réduction de leur volume.

Le Chapitre 3 montre que les approches proposées pour la réduction des traces d'exécution reposent principalement soit sur l'utilisation des algorithmes de compression tels que *LZ78*, soit sur l'identification de la redondance dans les données, pour fournir une représentation compacte ou compressée des données dans les traces d'exécution. Les traces réduites issues de ces approches sont inadaptées ou ne disposent d'aucun intérêt pour l'analyse d'applications, notamment pour le débogage ou le profilage.

Face à l'inadéquation des approches de réduction des traces d'exécution pour l'analyse d'applications nous avons proposé, au chapitre 4, une approche de réduction des trace d'exécution qui repose principalement sur la détection des anomalies. Notre approche de réduction utilise la distribution des événements dans les traces d'exécution pour modéliser les régimes d'exécution d'une application et identifier les régimes réguliers et les régimes irréguliers. Elle comporte deux phases : une phase d'apprentissage qui consiste à déterminer les régimes d'exécution réguliers sans connaissance a priori de l'application, et une phase de détection d'anomalies qui utilise une approche de détection d'anomalies de l'état de l'art (approche d'identification des anomalies locales basées sur la densité) pour détecter les régimes irréguliers. Les régimes irréguliers sont des régimes qui diffèrent des régimes réguliers. La trace réduite obtenue par notre approche

de réduction est constituée uniquement des parties de la trace d'exécution complète décrites par les régimes irréguliers.

L'étude expérimentale présentée au Chapitre 5 nous a permis d'évaluer différents aspects de notre approche de réduction. Il se dégage que notre approche de réduction présente en moyenne une précision de 82% pour un rappel de 98.2%, ce qui montre que notre approche de réduction détecte effectivement les comportements anormaux d'une application et que la trace d'exécution réduite est globalement constituée des périodes d'exécution correspondant à ces comportements anormaux. Notre approche de réduction permet d'obtenir une performance de réduction de données d'un ordre de grandeur, ce qui la présente comme une contribution importante pour le passage à l'échelle des approches d'analyses de traces post-mortem sur des grands volumes de traces d'exécution. Par ailleurs, notre approche présente un débit de traitement pouvant atteindre 21.8 Mbits/s. Ce qui représente un débit de traitement suffisamment important pour suivre les débits de production des traces des systèmes embarquées tels que les boîtes de capture des Set-Top Box pour lesquelles le débit réel de génération de traces d'exécution est de 12 Mbits/s en moyenne.

Quelques Perspectives

L'analyse des traces d'exécution devient un domaine de recherche de plus en plus actif à cause des divers types d'applications auxquels elle s'applique et s'adaptent. Notre travail se présente comme l'introduction d'une nouvelle approche de réduction des données qui porte les instances d'observation qui diffèrent des tendances générales ou attendues. À cet effet, plusieurs perspectives se dégagent de notre approche tant pour la réduction des données des traces d'exécution que pour l'interprétabilité des données réduites.

Intégration de notre approche dans les équipements de trace

L'approche de réduction de trace que nous proposons vise à permettre aux outils d'analyse de trace d'exécution de passer à l'échelle sur des grands volumes des traces d'exécution. Dans la chaîne d'analyse d'application à partir des traces d'exécution, notre approche se situe entre la génération des traces d'exécution et l'analyse des traces d'exécution. Il serait de ce fait intéressant de l'intégrer directement dans les boîtes de capture de trace développées par STMicroelectronics. Cette intégration permettrait de récupérer immédiatement les parties critiques de la trace d'exécution pour l'analyse post-mortem.

Interprétation et détection

– Notre approche de réduction de traces d'exécution fournit des indications sur l'irrégularité d'un régime en termes de déviation par rapport aux différents types d'événements. Par exemple, le régime R est irrégulier parce que la fenêtre d'événements qu'il décrit présente un grand nombre d'événements de type e_1 par rapport au nombre d'événements de type e_1 de chacune des fenêtres d'événements dont le régime est proche. Ces indications bien que pertinentes constituent une interprétation basique des irrégularités des régimes. Une forme d'indication sémantique plus intéressante consisterait à décrire l'irrégularité d'un régime en termes de déviation par rapport au(x) régime(s) régulier(s) duquel (desquels) ce régime dévie. Ce type d'indication sémantique fournirait une interprétation plus pertinente des comportements associés aux régimes irréguliers.

– La trace réduite produite par notre approche est une trace au format de la trace de base c'est à dire une séquence d'événements horodatés. Notre objectif est de faciliter l'utilisation des outils d'analyse de traces post-mortem mais aussi de guider le développeur dans l'analyse de l'application. Une perspective consisterait à construire des métadonnées à partir des informations caractéristiques des régimes irréguliers (par exemple des motifs) pour annoter les parties de la trace réduite dans le but de faciliter la compréhension de la trace réduite et accélérer le travail du développeur.

Réduction des données

L'approche que nous proposons produit en résultat une trace d'exécution réduite constituée uniquement des parties de la trace de base qui sont associées à des périodes d'exécution problématiques d'une application. Bien que réduite, la trace d'exécution peut contenir de la redondance d'information provenant soit des périodes d'exécution associées à une anomalie répétitive, soit des périodes d'exécution associées aux anomalies présentant des caractéristiques communes. Une perspective de réduction de la trace produite par notre approche consisterait à exprimer les caractéristiques (données) des périodes d'exécution anormales identifiées sous forme de motifs (itemsets, séquences, périodiques, ...) afin de déterminer les motifs fréquents dans la trace réduite. Ainsi, il serait intéressant de trouver comment coupler le principe de réduction MDL [VVLS11] avec les motifs fréquents extraits de manière à mettre en évidence les caractéristiques (motifs) communes aux périodes d'exécution problématiques, et de regrouper ces périodes d'exécution problématiques en fonction de ces caractéristiques communes.

Bibliographie

[ADFdB13] Azzeddine Amiar, Mickaël Delahaye, Yliès Falcone, and Lydie du Bousquet. Compressing microcontroller execution traces to assist system analysis. In Gunar Schirner, Marcelo Götz, Achim Rettberg, MauroC. Zanella, and FranzJ. Rammig, editors, *Embedded Systems : Design, Analysis and Verification*, volume 403 of *IFIP Advances in Information and Communication Technology*, pages 139–150. Springer Berlin Heidelberg, 2013.

[AFV95] Debra Anderson, Thane Frivold, and Alfonso Valdes. *Next-generation intrusion detection expert system (NIDES) : A summary*. SRI International, Computer Science Laboratory, 1995.

[Ami13] Azzeddine Amiar. *Assit to execution trace analysis in the microcontrollers 32 bits context*. Theses, Université de Grenoble, November 2013.

[AOC07] Tarem Ahmed, Boris Oreshkin, and Mark Coates. Machine learning approaches to network anomaly detection. In *Proceedings of the 2Nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, SYSML'07, pages 7 :1–7 :6, Berkeley, CA, USA, 2007. USENIX Association.

[AP02] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *Principles of Data Mining and Knowledge Discovery, 6th European Conference, PKDD 2002, Helsinki, Finland, August 19-23, 2002, Proceedings*, pages 15–26, 2002.

[Aug83] Stan Augarten. *State of the Art : A Photographic History of the Integrated Circuit*. Houghton Mifflin, 1983.

[AY01] Charu C Aggarwal and Philip S Yu. Outlier detection for high dimensional data. In *ACM Sigmod Record*, volume 30, pages 37–46. ACM, 2001.

[BC04] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Springer, 2004.

[BDRL05] Emmanuel Bouix, Marc Dalmau, Philippe Roose, and Franck Luthon. Multimedia oriented component model. In *19th International Conference on Advanced Information Networking and Applications (AINA 2005), 28-30 March 2005, Taipei, Taiwan*, pages 3–8, 2005.

[BGH06] George K. Baah, Alexander G. Gray, and Mary Jean Harrold. On-line anomaly detection of deployed software : a statistical machine learning approach. In *Third International Workshop on Software Quality Assurance, SOQUA 2006, Portland, Oregon, USA, November 6, 2006*, pages 70–77, 2006.

[BGRS99] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful ? In *Database Theory—ICDT'99*, pages 217–235. Springer, 1999.

[BHJM09] Dirk Beyer, Thomas A. Henzinger, Ranjit. Jhala, and Rupak Majumdar. The software model checker blast : Applications to software engineering. 2009.

[BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF : identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 93–104, 2000.

[BMR10] Endri Bezati, Marco Mattavelli, and Mickaël Raulet. RVC-CAL dataflow implementations of MPEG AVC/H.264 CABAC decoding. In *Proceedings of the 2010 Conference on Design & Architectures for Signal & Image Processing, DASIP 2010, Edinburgh, Scotland, UK, October 26-28, 2010, Electronic Chips & Systems design Initiative, ECSI*, pages 207–213, 2010.

[BR03] J. Bilek and I.P. Ruzicka. Evolutionary trends of embedded systems. In *Industrial Technology, 2003 IEEE International Conference on*, volume 2, pages 901–905 Vol.2, Dec 2003.

[BS05] R. P. Jagadeesh Chandra Bose and S. H. Srinivasan. Data mining approaches to software fault diagnosis. In *15th International Workshop on Research Issues in Data Engineering (RIDE-SDMA 2005), Stream Data Mining and Applications, 3-7 April 2005, Tokyo, Japan*, pages 45–52, 2005.

[BWJ01] Daniel Barbara, Ningning Wu, and Sushil Jajodia. Detecting novel network intrusions using bayes estimators. In *SDM*, pages 1–17. SIAM, 2001.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection : A survey. *ACM Comput. Surv.*, 41(3), 2009.

[CBT+12] Patricia López Cueva, Aurélie Bertaux, Alexandre Termier, Jean-François Méhaut, and Miguel Santana. Debugging embedded multimedia application traces through periodic pattern mining. In *Proceedings of the 12th International Conference on Embedded Software, EMSOFT 2012, part of the Eighth Embedded Systems Week, ESWeek 2012, Tampere, Finland, October 7-12, 2012*, pages 13–22, 2012.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252, 1977.

[CCCH09] Rung Ching Chen, Kai-Fan Cheng, Ying-Hao Chen, and Chia-Fen Hsieh. Using rough set and support vector machine for network intrusion detection system. In *First Asian Conference on Intelligent Information and Database Systems, ACIIDS 2009, Dong hoi, Quang binh, Vietnam, April 1-3, 2009*, pages 465–470, 2009.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263, 1986.

[CL13] Robert Christensen and Feifei Li. Adaptive log compression for massive log data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1283–1284, 2013.

[CS06] Sanjay Chawla and Pei Sun. Slom : a new measure for local spatial outliers. *Knowledge and Information Systems*, 9(4) :412–429, 2006.

[CZH+08] Bas Cornelissen, Andy Zaidman, Danny Holten, Leon Moonen, Arie van Deursen, and Jarke J. van Wijk. Execution trace analysis through massive sequence and circular bundle views. *Journal of Systems and Software*, 81(12) :2252–2268, 2008.

[DGIM02a] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6) :1794–1813, 2002.

[DGIM02b] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM J. Comput.*, 31(6) :1794–1813, 2002.

[DKVY06] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*, 2006.

[DL97] Manoranjan Dash and Huan Liu. Feature selection for classification. *Intell. Data Anal.*, 1(1-4) :131–156, 1997.

[DN00] Dipankar Dasgupta and Fernando Nino. A comparison of negative and positive selection algorithms in novel pattern detection. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 1, pages 125–130. IEEE, 2000.

[EAP+02] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer, 2002.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231, 1996.

[ESK03] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1-3, 2003*, pages 47–58, 2003.

[Evr13] Atif Ahmet Evren. Some applications of kullback-leibler and jeffreys' divergences in multinomial populations. *Journal of Selcuk University Natural and Applied Science*, 1(4) :pp–48, 2013.

[FR+12] Stephen E Fienberg, Alessandro Rinaldo, et al. Maximum likelihood estimation in log-linear models. *The Annals of Statistics*, 40(2) :996–1023, 2012.

[FSS90] Philippe Flajolet, Paolo Sipala, and Jean-Marc Steyaert. Analytic variations on the common subexpression problem. In *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, pages 220–234, 1990.

[GMES+99] Sigurour E Guttormsson, RJ Marks, MA El-Sharkawi, I Kerszenbaum, et al. Elliptical novelty grouping for on-line short-turn detection of excited running rotors. *Energy Conversion, IEEE Transactions on*, 14(1) :16–22, 1999.

[GSt15] GStreamer. Gstreamer debugging tool, 2015. Accédé le 7 janvier 2015.

[GZK05] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams : a review. volume 34, pages 18–26. ACM, 2005.

[Has97] Barry G Haskell. *Digital Video : An Introduction to MPEG-2 : An Introduction to MPEG-2*. Springer, 1997.

[HBK+03] Kimmo Hätönen, Jean-François Boulicaut, Mika Klemettinen, Markus Miettinen, and Cyrille Masson. Comprehensive log compression with frequent patterns. In *Data Warehousing and*

Knowledge Discovery, 5th International Conference, DaWaK 2003, Prague, Czech Republic, September 3-5,2003, Proceedings, pages 360–370, 2003.

[HD80] Hawkins and M. Douglas. *Identification of outliers*, volume 11. Springer, 1980.

[HFLP01] Paul S Horn, Lan Feng, Yanmei Li, and Amadeo J Pesce. Effect of outliers and nonhealthy individuals on reference interval estimation. *Clinical Chemistry*, 47(12) :2137–2145, 2001.

[HKF04] Ville Hautamäki, Ismo Kärkkäinen, and Pasi Fränti. Outlier detection using k-nearest neighbour graph. In *ICPR (3)*, pages 430–433, 2004.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining : Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[HLBAL05] A. Hamou-Lhadj, E. Braun, Daniel Amyot, and T. Lethbridge. Recovering behavioral design models from execution traces. In *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on*, pages 112–121, March 2005.

[HLL02] Abdelwahab Hamou-Lhadj and Timothy C Lethbridge. Compression techniques to simplify the analysis of large execution traces. In *Program Comprehension, 2002. Proceedings. 10th International Workshop on*, pages 159–168. IEEE, 2002.

[HLL04] Abdelwahab Hamou-Lhadj and Timothy C Lethbridge. A survey of trace exploration tools and techniques. In *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pages 42–55. IBM Press, 2004.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10) :576–580, oct 1969.

[Huf52] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9) :1098–1101, Sept 1952.

[HXD03] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recogn. Lett.*, 24(9-10) :1641–1650, june 2003.

[IBJ+14] K.E. Isaacs, P.-T. Bremer, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, and B. Hamann. Combing the communication hairball : Visualizing parallel execution traces using logical time. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12) :2349–2358, Dec 2014.

[JCJ14] Soo-Yeon Ji, Seonho Choi, and Dong Hyun Jeong. Designing a two-level monitoring method to detect network abnormal behaviors. In *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*, pages 703–709, Aug 2014.

[Jef61] H Jeffreys. Theory of probability, 1961.

[KBD+08] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S Müller, and Wolfgang E Nagel. The vampir performance analysis toolset. In *Tools for High Performance Computing*, pages 139–155. Springer Berlin Heidelberg, 2008.

[KBDG04] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 180–191. VLDB Endowment, 2004.

[KG06] A. Kuhn and O. Greevy. Exploiting the analogy between traces and signal processing. In *Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference on*, pages 320–329, Sept 2006.

[KIRT13] Christiane Kamdem Kengne, Noha Ibrahim, Marie-Christine Rousset, and Maurice Tchuente. Distance-based trace diagnosis for multimedia applications : Help me ted ! In *2013 IEEE Seventh International Conference on Semantic Computing, Irvine, CA, USA, September 16-18, 2013*, pages 306–309, 2013.

[KL51] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, pages 79–86, 1951.

[KMM00] Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-aided reasoning : an approach*. Kluwer Academic Publishers, 2000.

[KNT00] Edwin M. Knorr, Raymond T. Ng, and V. Tucakov. Distance-based outliers : Algorithms and applications. *VLDB J.*, 8(3-4) :237–253, 2000.

[KS05] Youngsoo Kim and Suleyman Sair. Designing real-time h.264 decoders with dataflow architectures. In *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '05, pages 291–296, New York, NY, USA, 2005. ACM.

[KSD09] A. Kind, M.P. Stoecklin, and X. Dimitropoulos. Histogram-based traffic anomaly detection. *Network and Service Management, IEEE Transactions on*, 6(2) :110–121, June 2009.

[KWK10] Johan Kraft, Anders Wall, and Holger M. Kienle. Trace recording for embedded systems : Lessons learned from five industrial projects. In Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann, editors, *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, volume 6418 of *Lecture Notes in Computer Science*, pages 315–329. Springer, 2010.

[LC13] Patricia Lopez Cueva. *Debugging Embedded Multimedia Application Execution Traces through Periodic Pattern Mining*. Theses, Université de Grenoble, July 2013.

[LK05] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166. ACM, 2005.

[LLP07] LonginJan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. Outlier detection with kernel density functions. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, volume 4571 of *Lecture Notes in Computer Science*, pages 61–75. Springer Berlin Heidelberg, 2007.

[LTP14] Sofiane Lagraa, Alexandre Termier, and Frédéric Pétrot. Scalability bottlenecks discovery in mpsoc platforms using data mining on simulation traces. In *DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6, 2014.

[LV02] Khaled Labib and Rao Vemuri. Nsom : A real-time network-based intrusion detection system using self-organizing maps. *Networks and Security*, pages 1–6, 2002.

[LXY03] Jianjiang Lu, Baowen Xu, and Hongji Yang. Matrix dimensionality reduction for mining web logs. In *2003 IEEE / WIC International Conference on Web Intelligence, (WI 2003), 13-17 October 2003, Halifax, Canada*, pages 405–408, 2003.

[LYY+05] Chao Liu, Xifeng Yan, Hwanjo Yu, Jiawei Han, and Philip S. Yu. Mining behavior graphs for "backtrace" of noncrashing bugs. In *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005, Newport Beach, CA, USA, April 21-23, 2005*, pages 286–297, 2005.

[Mah36] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2 :49–55, 1936.

[MM63] Joan C. Miller and Clifford J. Maloney. Systematic mistake analysis of digital computer programs. *Communications of the ACM*, 6(2) :58–63, 1963.

[MM03] Aleksandar Milenkovic and Milena Milenkovic. Exploiting streams in instruction and data address trace compression. In *Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on*, pages 99–107. IEEE, 2003.

[MSB11] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[Mur12] Richard Murch. *The Software Development Lifecycle - A Complete Guide*. Richard Murch, 2012.

[Mye04] Glenford J. Myers. *The art of software testing (2. ed.)*. Wiley, 2004.

[NKNW96] John Neter, Michael H Kutner, Christopher J Nachtsheim, and William Wasserman. *Applied linear statistical models*, volume 4. Irwin Chicago, 1996.

[NMW97] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences : A linear-time algorithm. *J. Artif. Int. Res.*, 7(1) :67–82, September 1997.

[ORS92] Sam Owre, John M. Rushby, and Natarajan Shankar. Pvs : A prototype verification system. In *Automated Deduction—CADE-11*, pages 748–752. Springer, 1992.

[PKGF03] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. LOCI : fast outlier detection using the local correlation integral. In *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 315–326, 2003.

[PLL07] D. Pokrajac, A. Lazarevic, and L.J. Latecki. Incremental local outlier detection for data streams. In *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*, pages 504–515, March 2007.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57, 1977.

[POH09] Hae-woo Park, Hyunok Oh, and Soonhoi Ha. Multiprocessor soc design methods and tools. *Signal Processing Magazine, IEEE*, 26(6) :72–79, 2009.

[Poo97] Viswanath Poosala. *Histogram-based Estimation Techniques in Database Systems*. PhD thesis, Madison, WI, USA, 1997. UMI Order No. GAX97-16074.

[Pou14] Kevin Pouget. *Programming-Model Centric Debugging for multicore embedded systems*. Theses, Université de Grenoble, February 2014.

[PP07] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques : Existing solutions and latest technological trends. *Computer Networks*, 51(12) :3448 – 3470, 2007.

[PPB+06] Erez Perelman, Marzia Polito, J.-Y. Bouguet, Jack Sampson, Brad Calder, and Carole Dulong. Detecting phases in parallel applications on shared memory architectures. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10–pp. IEEE, 2006.

[PR11] Prada rojas Prada Rojas, Carlos Hernan. *A component-based observation approach for MP-SoC observation*. Theses, Université de Grenoble, June 2011.

[PTVF07] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. Numerical recipes : the art of scientific computing. 3rd, 2007.

[Qua15] Qualcomm. Processeurs snapdragon 801, 2013. Accédé le 11 février 2015.

[RHL+08] Fei Ren, Liang Hu, Hao Liang, Xiaobo Liu, and Weiwu Ren. Using density-based incremental clustering for anomaly detection. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 3, pages 986–989. IEEE, 2008.

[RMSM02] Gunnar Rätsch, Sebastian Mika, Bernhard Schölkopf, and Klaus-Robert Müller. Constructing boosting algorithms from svms : An application to one-class classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(9) :1184–1199, 2002.

[ROT03] Manikantan Ramadas, Shawn Ostermann, and Brett C. Tjaden. Detecting anomalous network traffic with self-organizing maps. In *Recent Advances in Intrusion Detection, 6th International Symposium, RAID 2003, Pittsburgh, PA, USA, September 8-10, 2003, Proceedings*, pages 36–54, 2003.

[SC04] Pei Sun and Sanjay Chawla. On local spatial outliers. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 209–216. IEEE, 2004.

[SS07] Przemysław Skibiński and Jakub Swacha. Fast and efficient log file compression. In *CEUR 0Workshop Proceedings of 11th East-European Conference on Advances in Databases and Information Systems (ADBIS 2007)*, 2007.

[STM] STMicroelectronics. Stworkbench.

[SW07] M Sheldon and Ganesh Venkitachalam Boris Weissman. Retrace : Collecting execution trace with virtual machine deterministic replay. In *Proceedings of the Third Annual Workshop on Modeling, Benchmarking and Simulation (MoBS 2007)*, 2007.

[SWZK12] Erich Schubert, Remigius Wojdanowski, Arthur Zimek, and Hans-Peter Kriegel. On evaluation of outlier rankings and outlier scores. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012.*, pages 1047–1058, 2012.

[TAB+05] Chinh Tran, Chijioke Anyanwu, Sanjai Balakrishnan, Anshul Bhargava, James Jiang, and Radhika Thekkath. The mips32® 24ke™ core family : High-performance risc cores with dsp enhancements. 2005.

[TBS+11] Lionel Torres, Pascal Benoit, Gilles Sassatelli, Michel Robert, Fabien Clermidy, and Diego Puschini. An introduction to multi-core system on chip – trends and challenges. In Michael Hübner and Jürgen Becker, editors, *Multiprocessor System-on-Chip*, pages 1–21. Springer New York, 2011.

[TCFC02] Jian Tang, Zhixiang Chen, Ada Wai-Chee Fu, and David W Cheung. Enhancing effectiveness of outlier detections for low density patterns. In *Advances in Knowledge Discovery and Data Mining*, pages 535–548. Springer, 2002.

[Tud95] P.N. Tudor. Mpeg-2 video compression. *Electronics & Communication Engineering Journal*, 7 :257–264(7), December 1995.

[VN12] V Vijayakumar and R Nedunchezhian. A study on video data mining. *International journal of multimedia information retrieval*, 1(3) :153–172, 2012.

[VVLS11] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. Krimp : mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1) :169–214, 2011.

[Wil03] James M. Wilson. Gantt charts : A centenary appreciation. *European Journal of Operational Research*, 149(2) :430–437, 2003.

[WJM08] Wayne Wolf, Ahmed Amine Jerraya, and Grant Martin. Multiprocessor system-on-chip (mpsoc) technology. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(10) :1701–1713, 2008.

[WWZ+14] Tao Wang, Jun Wei, Wenbo Zhang, Hua Zhong, and Tao Huang. Workload-aware anomaly detection for web applications. *Journal of Systems and Software*, 89(0) :19 – 32, 2014.

[YQLZ06] Jeffrey Xu Yu, Weining Qian, Hongjun Lu, and Aoying Zhou. Finding centric local outliers in categorical/numerical spaces. *Knowledge and Information Systems*, 9(3) :309–338, 2006.

[ZBY+12] Wenke Zhang, Favyen Bastani, I-Ling Yen, Kevin Hulin, Farokh B. Bastani, and Latifur Khan. Real-time anomaly detection in streams of execution traces. In *14th International IEEE Symposium on High-Assurance Systems Engineering, HASE 2012, Omaha, NE, USA, October 25-27, 2012*, pages 32–39, 2012.

[ZD04] Andy Zaidman and Serge Demeyer. Managing trace data volume through a heuristical clustering process based on event execution frequency. In *8th European Conference on Software Maintenance and Reengineering (CSMR 2004), 24-26 March 2004, Tampere, Finland, Proceedings*, pages 329–338, 2004.

[ZL78] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5) :530–536, 1978.

[ZSK+14] Mohammad Mehdi Zeinali Zadeh, Mahmoud Salem, Neeraj Kumar, Greta Cutulenco, and Sebastian Fischmeister. Sipta : Signal processing for trace-based anomaly detection. In *Proceedings of the 14th International Conference on Embedded Software*, EMSOFT '14, pages 6 :1–6 :10, New York, NY, USA, 2014. ACM.