# Analyse de traces d'exécutions pour les systèmes embarqués : détection d'anomalies par corrélation temporelle

Alexis Martin, Vania Marangozova-Martin


# Analyse de traces d'exécutions pour les systèmes embarqués : détection d'anomalies par corrélation temporelle

**Alexis Martin, Vania Marangozova-Martin**

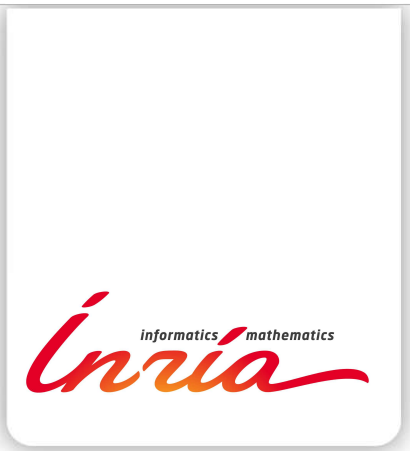
**TECHNICAL REPORT**

**N° 450**

Octobre 2014

Project-Team Mescal

ISSN 0249-0803 ISRN INRIA/RT--450--FR+ENG



# Analyse de traces d'exécutions pour les systèmes embarqués : détection d'anomalies par corrélation temporelle

Alexis Martin [*] [†], Vania Marangozova-Martin [*] [‡]

Équipe-Projet Mescal

Rapport technique n° 450 — Octobre 2014 — 20 pages

**Résumé :** La complexité croissante des systèmes embarqués, tant au niveau matériel que logiciel, rend difficile leur développement et en particulier les tâches de débogage et d'optimisation des performances. L'utilisation d'outils de débogage ou d'analyse interactifs implique une perturbation importante du fonctionnement normal des systèmes. L'alternative est d'effectuer la mise au point du système après son exécution en utilisant des traces d'exécution.
Les traces d'exécution de systèmes embarqués contiennent usuellement de nombreuses informations concernant des événements de bas niveau (appels système, interruptions,...). Elles ne donnent qu'une vision microscopique du comportement et sont difficiles à comprendre et à analyser pour un acteur humain. Les développeurs ont besoin d'outils qui représentent les traces de manière synthétique et qui les aiguillent dans leur recherche de problèmes.
Dans ce rapport nous décrivons une méthode de detection automatique de problèmes d'exécution se basant sur des traitements statistiques et la notion de correlation temporelle. La méthode distingue entre phénomènes normaux et anormaux dans la trace et identifie des possibles déclencheurs de ces derniers. La méthode a été implémentée au sein de l'infrastructure FrameSoC [12] et a été validée avec un scénario de décodage multimédia.

**Mots-clés :** trace d'exécution, analyse, statistiques, correlation temporelle

[*] Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France / Inria / CNRS, LIG, F-38000 Grenoble, France
[†] alexis.martin@inria.fr
[‡] vania.marangozova-martin@imag.fr

**RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES**

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

**Abstract:** The increasing hardware and software complexity makes the debugging and the optimization of embedded systems a challenging task. In this context, interactive debugging is hindered by the system scale and the perturbation it incurs. The alternative is to analyze system behavior *a posteriori* using execution traces.

Trace exploitation, however, needs to tackle the issues of analyzing important volumes of low-level data (e.g. system calls, interruptions...). Indeed, traces do not reflect application semantics and provide a microscopic vision of the system execution which is difficult to understand by developers. Trace exploitation needs to be supported by tools capable of representing traces in synthetic way and of pointing out possible problematic zones.

In this report we investigate automatic bug detection. Using statistics, we distinguish between normal and abnormal events and investigate possible problem causes using temporal correlation. We have implemented our approach in the FrameSoC framework [12] and have validated it using a multimedia decoding scenario.

**Key-words:** execution trace, analysis, statistics, temporal correlation

# 1 Introduction

Le développement des télécommunications mobiles, l'utilisation d'objets intelligents (*smart objects*) ou l'Internet des objets (*IoT*) ont contribué à l'exploitation généralisée ces dernières années des systèmes embarqués dans de nombreux domaines de notre vie.

Cette nouvelle dynamique du marché et ses nombreuses applications imaginées exigent des temps de production raccourcis tout en maintenant les contraintes de calcul temps-réel, d'autonomie, de faible consommation énergétique et de robustesse. Dans ce contexte, les tâches de mise au point du logiciel embarqué deviennent critiques.

La mise au point du logiciel embarqué peut être fait *a priori*, avant l'exécution, ou *a posteriori*, après l'exécution. La première approche est basée sur la modélisation de la plate-forme matérielle et des traitements logiciels. La conception est guidée par les contraintes et garantit leur respect dans le produit final. La complexité croissante des systèmes embarqués et leur hétérogénéité rendent toutefois difficile l'application de cette méthode dans le cas général. L'approche alternative consiste à analyser le comportement du système en récupérant des informations sur son exécution. Ces informations peuvent donner une estimation globale sur les performances du système (*profiling*) ou contenir un historique détaillé d'exécution (*traçage*). Si le profiling peut donner des indications sur l'existence de problèmes à l'exécution, c'est le traçage qui fournit un historique détaillé qui permet de les identifier.

L'utilisation de traces a plusieurs avantages. De point de vue débogage, le traçage est bien moins intrusif qu'un débogage interactif (comme avec GDB) qui modifie facilement le cheminement d'exécution. D'autre part, la sauvegarde persistante des traces permet d'appliquer différents type d'analyses et ce sur la même execution. Cela évite les problèmes de non déterminisme et la difficulté de reproduction d'une exécution.

L'exploitation de traces reste confrontée, néanmoins, à des problèmes de volume de données et de manque de méthodes d'analyse évoluées. En effet, à titre d'exemple, un décodage vidéo de quelques secondes peut produire un trace de plusieurs mega octets et facilement atteindre les gigaoctets. Ce type de trace contiendrait typiquement des milliers, voir des millions d'événements reflétant l'exécution des couches basses du système. L'exploitation usuelle est de représenter la trace visuellement en laissant l'exploration et l'analyse aux développeurs expérimentés. Les environnements existants [17, 8] manquent d'outils qui permettent d'aiguiller le travail de mise au point en automatisant la recherche de problèmes.

Le projet SoC-Trace [3, 12] est un projet du Ministère de l'Industrie (projet FUI) avec comme partenaires l'Université Joseph Fourier, Inria Grenoble et les sociétés STMicroelectronics, ProBayes et Magillem Design Services. SoC-Trace a pour objectif le développement d'une infrastructure logicielle pour l'exploitation de traces d'exécution de systèmes embarqués. Cette infrastructure doit permettre un cycle d'exploitation de traces complet, capable de prendre en charge l'ensemble du flot de traces d'exécution en fournissant des services de stockage, d'accès optimisé et d'analyse comprenant la structuration d'informations, la visualisation efficace ou l'agrégation.

Dans le contexte du projet SoC-TRACE, nous proposons une méthode de détection automatique d'anomalies dans une trace d'exécution pour ainsi donner des points d'entrée au développeur pour une analyse plus ciblée. La méthode détecte des anomalies en utilisant des mesures statistiques simples et identifie des causes probables en établissant des correlations temporelles entre les informations tracées. La méthode a d'abord été explorée de manière manuelle en utilisant l'outil R et a ensuite été implémentée sous forme d'outil au sein de l'infrastructure FrameSoC développée dans le cadre du projet SoC-TRACE.

Ce rapport est organisé comme suit. La Section 2 positionne le problème de la recherche

d'anomalie dans une trace. La Section 3 présente la démarche expérimentale avec l'outil R. La Section 4 présente l'outil qui intègre notre approche au sein de l'infrastructure FrameSoC. En Section 5 nous discutons d'outils existant d'analyse de données et de leur applicabilité dans notre contexte. La Section 6 conclue sur les résultats et les perspectives de ce travail.

# 2 Détection d'anomalies dans les traces d'exécution

## 2.1 L'hétérogénéité des traces

Une *trace d'exécution* est un un historique de l'exécution d'un système. Elle comprend une série d'informations qui reflète l'ordre d'occurrence de différents phénomènes d'exécution, appelés *événements*. L'ordre des événements est typiquement donné par des estampilles situant les événements dans le temps. Les autres informations enregistrées reflètent la plateforme d'exécution, l'outil de traçage, le programme que l'on trace et la nature des phénomènes tracés.

Un exemple de ce que peut contenir une trace est donné à la Figure 1. L'extrait contient plusieurs lignes, chacune contenant les données relatives à un événement. La première information est l'estampille. Ainsi, la première ligne indique qu'il y a eu un appel à `sys_select` à $266047\mu s$. La deuxième ligne capture un changement de contexte (`__switch_to`) à $1018502\mu s$. Les autres informations contiennent, entre autres, l'identifiant du processus ( `1036 (sshd)` ou `1021 (flush-0:11)` et le numéro de processeur (1 ou 0).

```
...
266047,,,,sys_select,1036 (sshd),1,0x01a80e18,0x01a80dc8,0x00000000
1018502,,,,__switch_to,0,1021 (flush-0:11),0
1078487,,,,__switch_to,20 (kworker/0:1),0,0
1092491,1092499,8,8,Interrupt,Interrupt 168 (GIC  eth0),,0,,,,,,,
1092501,1092581,80,80,SoftIRQ,SoftIRQ (net_rx_action),,0,,,,,,,
...
```

FIGURE 1 – Extrait de trace KPTrace

Nous constatons une grande hétérogénéité dans les formats de traces et dans les outils existant pour leur manipulation [9]. Il manque de modèle général de trace qui définirait une sémantique standard sur les informations capturées.

## 2.2 Un modèle générique pour les traces

Dans le contexte du projet SoC-TRACE et de l'implémentation de l'infrastructure FrameSoC, nous avons proposé un modèle générique pour les données d'une trace à événements [12, 9, 10]. Ce modèle générique sert de socle de base pour l'implémentation d'outils pour le stockage, l'accès et l'analyse de traces. Nous utilisons la notions et la sémantique de ce modèle pour la définition et la détection automatique d'anomalies. Dans la suite, nous ne donnons que les éléments principaux du modèle. Pour plus de détails, le lecteur peut se référer aux publications citées.

Dans FrameSoC, une trace est une collection d'événements. Les événements ont des attributs communs, auxquels peuvent se rajouter des attributs spécifiques. Les attributs communs sont ceux qui visent à donner une structure et une sémantique commune à toutes les traces. Ils comprennent :

— *Estampille*
L'estampille donne l'information sur le moment d'occurrence d'in événement.
— *Type d'événement*
La notion de type d'événement répond au besoin de distinguer les différents phénomènes se produisant à l'exécution. Cette notion n'est pas définie de manière formelle et dépend du domaine métier de la trace. Usuellement, les événements considérés de même nature auront le même type avec possiblement d'autres informations spécifiques les distinguant. Par exemple, les appels système peuvent correspondre à un type d'événement, le différents événements étant différenciés par la fonction appelée, le moment de l'appel et le fil d'exécution effectuant l'appel.
— *Producteur*
Le producteur d'un événement reflète une notion de contexte d'apparition. Des producteurs typiques sont les processeurs, les processus, les composants logiciels, etc.
— *Catégorie*
Inspirés par les travaux B. Stein [16], les catégories rajoutent une dimension supplémentaire aux événements en nous permettant de distinguer des événements ponctuels, les états, les liens et les variables [10]. Les *événements ponctuels* se produisent à un instant dans le temps. Les *états* représentent des durées, comme les durées d'exécution des fonctions ou les états d'un processeur (actif/inactif). Les liens représentent une relation causale entre deux entités. Enfin, les variables représentent de valeurs relevées ou calculées comme l'évolution de l'utilisation mémoire ou encore des compteurs de performances.

## 2.3 Anomalies dans une trace

Une anomalie dans le comportement d'un système est tout comportement non attendu[4].

Nous considérons les cas d'analyse de traces d'exécution où il n'y a pas de spécifications préalables disponibles sur le comportement attendu du système. Par conséquent, la détection automatique d'anomalies est faite en deux phases : la définition de comportements corrects (assimilés à des comportements attendus) du système d'abord et la détection des comportements divergents (anomalies) ensuite. Pour que l'identification d'un comportement correct soit possible, la trace doit contenir plusieurs enregistrements s'y référant. Cela est le cas, par exemple, des applications multimédia qui ont un comportement périodique.

Dans le cadre de FrameSoC, nous considérons les comportements attendus à granularité d'événement. En d'autres termes, nous nous intéressons à définir les événements attendus et détecter les événements représentant des anomalies. Les comportements normaux et divergents sont exprimés en fonction des valeurs des attributs communs des événements dans le modèle générique de trace présenté précédemment.

Dans ce travail, nous utilisons l'intervalle de confiance [14] pour estimer les comportements moyens, considérés normaux. L'intervalle de confiance à 99% pour une série de $n$ éléments et de variable $X$ est définie par :

$$\left[\bar{x} - 3 \times \frac{\sigma(X)}{\sqrt{n}}; \bar{x} + 3 \times \frac{\sigma(X)}{\sqrt{n}}\right]$$

Dans la formule, $\bar{x}$ est la moyenne des valeurs de la variable $X$ et $\sigma(X)$ est l'écart type.

Nous définissons comme anomalies les événements dont les valeurs des paramètres estimés sont en dehors de l'intervalle de confiance.

## 2.4 Causes d'une anomalie

Une anomalie peut se produire une seule fois pendant l'exécution ou apparaître plusieurs fois. Dans le cas d'une seule occurrence, nous laissons l'analyse au développeur. Dans le cas de plusieurs occurrences, nous nous intéressons à établir des liens de cause à effet qu'il pourrait y avoir entre ces occurrences et les autres événements de la trace. Pour cela, nous comparons l'ensemble d'occurrences d'événements anormaux du même type et des ensembles formées par les autres événements de la trace (regroupés également par type). Les questions auxquelles nous essayons de répondre sont les suivantes : *L'apparition d'une anomalie coïncide-t-elle avec l'apparition d'un autre événement ? L'anomalie apparaît-elle uniquement quand cet autre événement se produit ?*

Pour répondre à ces questions, nous utilisons le fait que les ensembles d'événements sont ordonnés dans le temps (série temporelle) et visons à détecter une *corrélation temporelle* entre deux séries d'événements. La première série contient les anomalies, alors que la deuxième contient l'ensemble des événements d'un autre type. Si une corrélation est établie, nous considérons que l'anomalie peut avoir comme cause probable les actions reflétées par la deuxième série d'événements.

Pour considérer deux séries d'événements, nous utilisons les *histogrammes*[13] et le *coefficient de corrélation linéaire*.

**Histogramme** Un histogramme est un outil statistique qui permet une visualisation par agrégation de valeurs sur un intervalle de temps. En coupant cet intervalle en plusieurs segments, on obtient une représentation plus synthétique de ces valeurs.

Dans notre cas, l'histogramme est utilisé pour représenter la densité (nombre d'occurrences) d'événements par intervalle de temps et par type d'événement. L'objectif et de pouvoir mettre en évidence les événements se produisant dans les mêmes intervalles de temps.

La difficulté ici est de déterminer en combien de segments nous allons découper notre espace de temps. En effet un trop petit nombre de segments ne permettrait pas de faire ressortir une concentration local d'événements. Il en est de même pour un nombre trop grand où l'on aurait une vision sur un trop petit intervalle de temps. Il est usuel de prendre comme nombre de segments : $\sqrt{N}$ où $N$ est le nombre d'éléments.

**Corrélation** La corrélation entre deux variables aléatoires est le degré de dépendance de ces deux variables [1]. Elle se mesure grâce au coefficient de corrélation linéaire qui est une mesure numérique comprise entre $-1$ et 1. Si la valeur absolue de cette métrique est comprise entre 0.5 et 1, on dit qu'il y a une corrélation forte. Sinon, il y a une faible corrélation.

Le coefficient de correlation est le rapport entre la covariance des deux variables aléatoires et le produit de leur écart-type :

$$corr(X, Y) = \frac{cov(X, Y)}{\sigma_X \times \sigma_Y}$$

En prenant comme variables les temps d'occurrence de deux types d'événements différents, l'utilisation de la corrélation nous permet d'établir si les événements se produisent en même temps [1]. De manière plus générale, il s'agit d'établir le degré de proximité temporelle entre les deux séries d'événements, ce qui peut être considéré comme une mesure de distance.

1. Dans notre cas, nous nous intéresserons aux événements se produisant dans les mêmes intervalles de temps.

# 3 Calcul de distance entre deux series d'événements

Dans cette partie nous présentons notre approche expérimentale avec le logiciel R [2]. La Section 3.1 présente les scénarios d'analyse de trace fournis par STMicroelectronics. La Section 3.2 présente l'intuition de corrélation entre deux ensembles d'événements en explorant la représentation visuelle. La Section 3.3 considère un premier calcul du coefficient de corrélation en découpant la trace en tranches de temps égales. La Section 3.4 considère le calcul du coefficient de corrélation dans le cas d'ensembles à forte disparité dans leur cardinalité. Le calcul est fait en découpant la trace en tranches de temps irréguliers. La Section 3.5 donne les schémas algorithmiques pour effectuer les deux calculs selon les méthodes présentées précédemment.

## 3.1 Cas d'usage

Nous disposons de deux traces fournies par STMicroelectronics. Les deux cas d'usage, nommés *Unicast* et *TSRecord*, concernent le fonctionnement d'un décodeur vidéo. Les traces brutes sont produites en traçant les différentes couches logicielles (e.g. noyau Linux, intergiciel ST, ...). Les informations tracées reflètent des appels de fonctions, des interruptions logicielles et matérielles, des communications réseau, etc.

### 3.1.1 Unicast

Dans le cas *Unicast*, le décodeur lit un fichier depuis le réseau et le diffuse sur sa sortie vidéo. Durant l'exécution surviennent des sauts d'images et des craquements audio qui indiquent un problème de décodage. Le problème de décodage se traduit dans la trace par un temps d'exécution trop long de certains appels à `SoftIRQ`. Ces durées anormalement longues sont en fait un effet de bord d'une autre tâche : `Flush`.

Dans le modèle FrameSoC, les événements `SoftIRQ` sont des événements de catégorie *état* et disposent de l'information de durée. Les anomalies sont les appels à `SoftIRQ` trop longs. Pour les identifier, nous considérons les durées des traitements `SoftIRQ`. et calculons leur moyenne $\bar{x}$ et l'écart type $\sigma$. Les éléments en dehors de l'intervalle de confiance sont considérés comme anormaux.

La figure 2 représente les événements `SoftIRQ` de la trace. Chaque point représente un événement avec en ordonnée son estampille et en abscisse sa durée. La partie (a) montre tous les événements, alors que la partie (b) contient uniquement les événements anormaux.

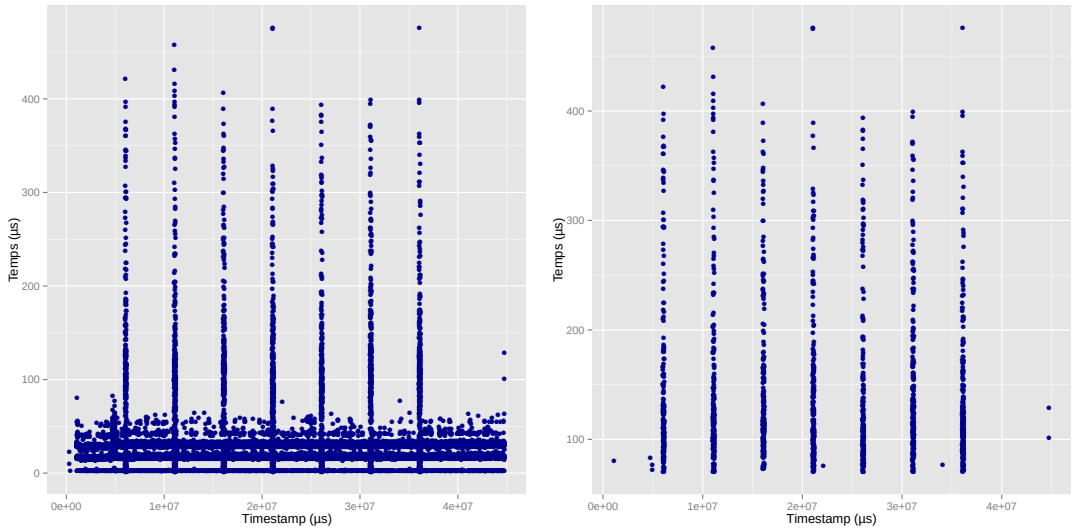
### 3.1.2 TSrecord

Dans le cas TSrecord, le décodeur lit un flux vidéo depuis le réseau et l'enregistre sur un disque dur USB. La vidéo enregistrée présente des sauts d'images et des craquements audio qui indiquent une perte d'information.

Le problème est le suivant. Toutes les $100ms$, les buffers IP qui stockent le flux vidéo sont vidés dans la zone mémoire correspondant au cache disque. Or, toutes les $5000ms$, le cache est verrouillé lors de son écriture sur disque. Par conséquent, le flux vidéo qui arrive entre temps est perdu.

Dans la trace, ceci peut être vu au niveau des appels à `TSrecord` qui est responsable de la lecture du flux vidéo depuis le réseau et plus particulièrement du vidage des buffers IP vers le

2. http ://www.r-project.org/




(a) Enssemble des événements `SoftIRQ` (b) Événements `SoftIRQ` anormaux

FIGURE 2 – Filtrage des événements `SoftIRQ` selon leur durée.

cache disque. `TSrecord` ne s'exécute pas lorsque survient la tâche `USB-storage`, responsable de l'écriture effective sur le disque.

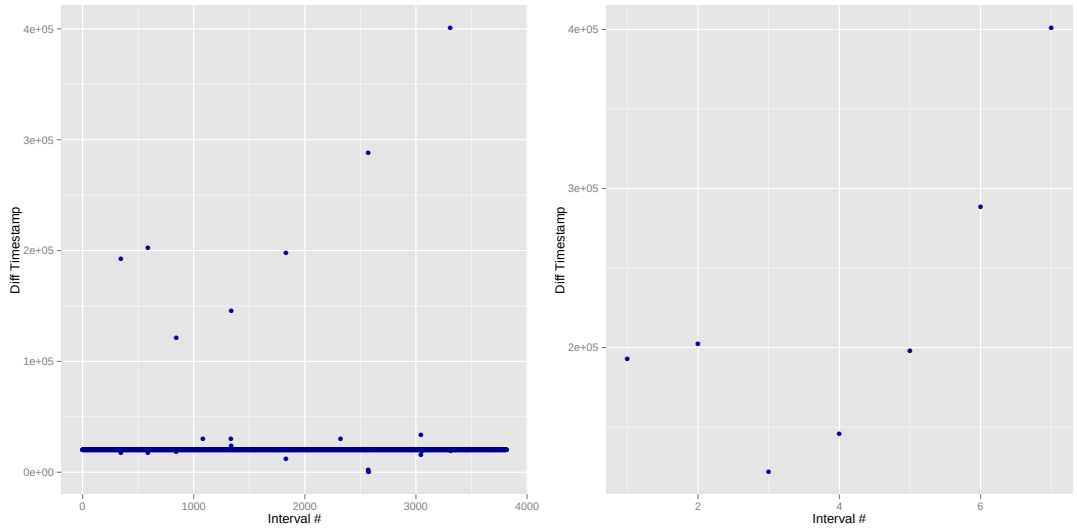
Dans ce cas d'usage nous nous intéressons à la périodicité des appels `TSrecord`. Pour cela nous considérons les intervalles de temps entre deux appels consécutifs. Nous considérons comme anormaux les éléments qui ne rentrent pas dans l'intervalle de confiance de la moyenne des durées des intervalles.

La figure 3 représente la périodicité des événements `TSrecord`. Chaque point représente la durée d'une intervalle entre deux appels consécutifs, avec en ordonnée le numéro de l'intervalle et en abscisse sa durée. La partie (a) montre tous les intervalles, alors que la partie (b) contient uniquement les intervalles anormalement longs.

## 3.2 Correlation visuelle

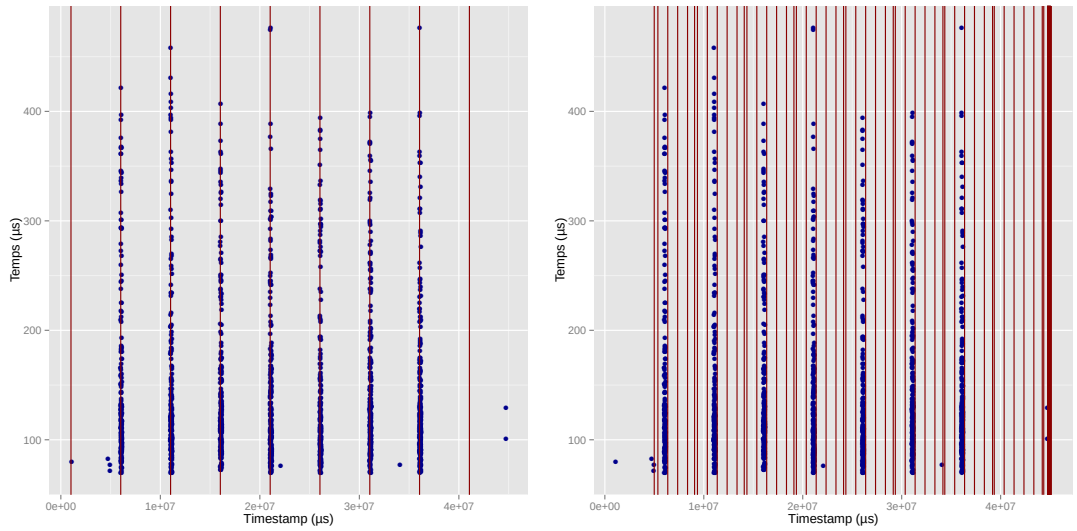
Dans le cas d'*Unicast*, nous superposons visuellement les événements `SoftIRQ` anormaux et les événements d'un autre type. Dans la figure 4, les événements `SoftIRQ` sont représentés par des points, alors que les les occurrences des autres événements sont indiqués par des traits rouges verticaux. Dans la partie (a) nous observons une superposition parfaite, alors que dans la partie (b) ce n'est pas le cas. En conclusion, les événements `Flush` sont intéressants à considérer en tant que cause probable des anomalies.

De manière analogue, nous pouvons établir visuellement un lien entre les événements `TSrecord` bloqués et les événements d'un autre type dans le cas d'usage *TSrecord*. Dans la figure 5, les événements `TSrecord` sont représentés par des traits bleu verticaux, alors que les les occurrences des autres événements sont indiqués par des points rouges. Dans la partie (a) nous observons une superposition majoritaire, alors que dans la partie (b) ce n'est pas le cas. En conclusion, les événements `USB-storage` sont intéressants à considérer en tant que cause probable des anomalies.




(a) Durée entre deux appels consécutifs à `TSrecord` (b) Intervalles à périodicité cassée

FIGURE 3 – Filtrage des événements `TSrecord` selon la durée entre deux appels consécutifs.

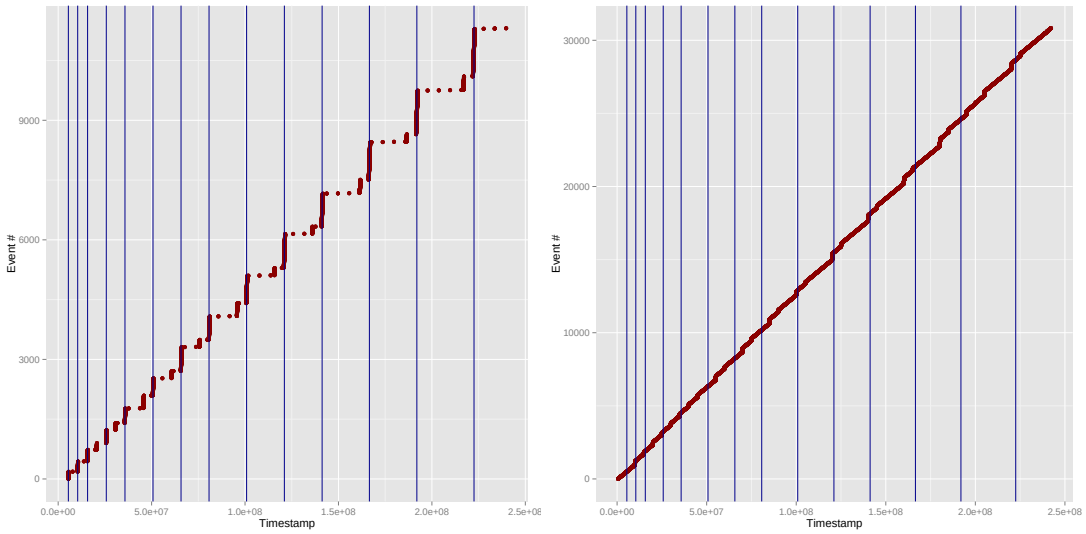



(a) *Unicast* : anomalies `SoftIRQ` superposés avec `Flush` (b) *Unicast* : anomalies `SoftIRQ` superposés avec `Cyclesoak`

FIGURE 4 – Correlation visuelle pour *Unicast*

## 3.3 Corrélation par découpe en tranches de temps égales

Notre objectif étant d'établir la corrélation entre deux ensembles d'événements de manière numérique et non visuelle, nous avons utilisé la mesure de corrélation linéaire appliquée à la




(a) Anomalies `TSrecord` superposés avec `USB-storage` (b) Anomalies `TSrecord` superposés avec `Kworker`

FIGURE 5 – Correlation visuelle pour *TSrecord*

densité d'événements des deux ensemble. Pour chaque ensemble, nous avons partagé l'exécution en tranches de temps égales et considérons le nombres d'occurrences par tranche. La densité pour chaque ensemble est donc représentée par une suite numérique dont chaque élément représente une tranche de temps. Nous utilisons la fonction de corrélation sur ces deux suites pour obtenir la valeur du coefficient de corrélation linéaire.

Pour le cas d'usage *Unicast* et les types d'événements représentés dans Figure 4, nous obtenons $0,85$ pour la corrélation entre `SoftIRQ` et `Flush` et $0,15$ pour la corrélation entre `SoftIRQ` et `Cyclesoak`. La mesure reflète donc bien ce que nous avons constaté de manière visuelle.

De manière analogue, pour le cas d'usage *TSrecord* et les types d'événements représentés dans Figure 5, nous obtenons $0,83$ pour la corrélation entre `TSrecord` et `USB-storage` et $0,11$ pour la corrélation entre `TSrecord` et `Kworker`.

## 3.4 Corrélation par découpe en tranches de temps irréguliers

Si nous explorons la corrélation entre un ensemble contenant quelques anomalies et un ensemble à plusieurs milliers d'événements, nous obtenons un graphique de corrélation très peu lisible (c.f Figure 6).

Pour effectuer une analyse plus représentative, nous utilisons les données sur les anomalies comme point de départ pour l'analyse et considérons les intervalles de temps aux alentours. Nous considérons les densités de points du deuxième ensemble dans les intervalles ainsi obtenus. La figure 7 montre un exemple de ce découpage avec en bleu les zones aux alentours des anomalies détectées. Ces intervalles sont définis selon un paramètre delta : $anomalie \pm delta$. La valeur de *delta* est déterminée par l'utilisateur. Une bonne valeur pour commencer l'analyse est environ 1% de l'espace de temps totale.

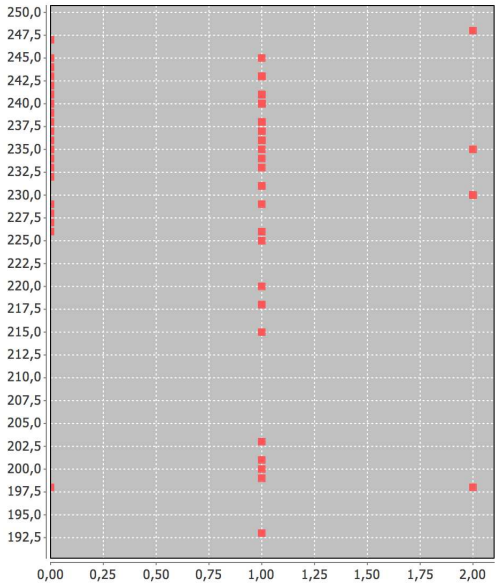



FIGURE 6 – Corrélation peu lisible due à une grande différence du nombre d'événements entre les deux ensembles.

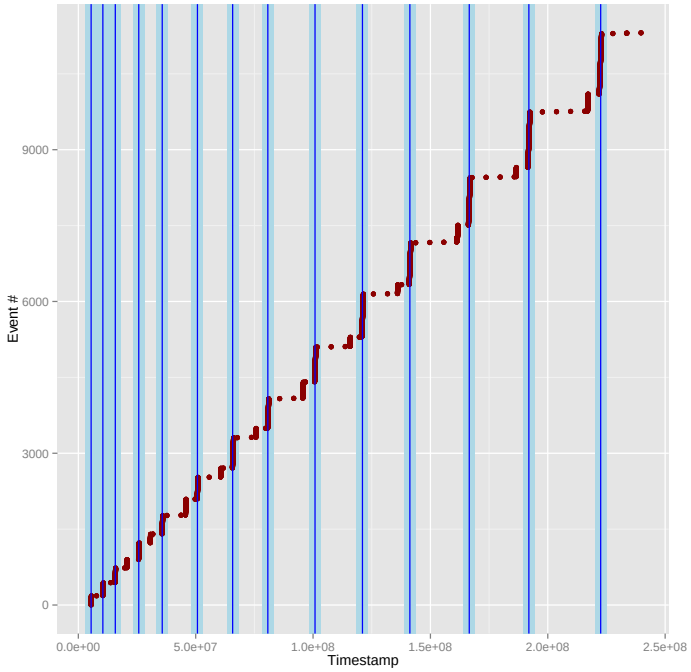



FIGURE 7 – En bleu, zones aux alentours des anomalies, elles-mêms représentées par des barres verticales.

### 3.5 Algorithmes

L'algorithme 1 présente la méthode de calcul de corrélation avec découpage en temps régulier.

---

**Data**:
L1 : liste des éléments de type 1 ;
L2 : liste des éléments de type 2 ;
**Result**:
Cor : Coefficient de corrélation linéaire ;

**if** *filtrer la liste L1 ?* **then**
| filtrer la liste L1 selon la catégorie ;
**end**
**if** *filtrer la liste L2 ?* **then**
| filtrer la liste L2 selon la catégorie ;
**end**
Prendre comme espace de temps l'union des espaces temps des deux listes;
Découper l'espace de temps en N tranches égales avec $N = sqrt(L1.size + l2.size)$;
Compter le nombre d'éléments de L1 par tranche de temps et sauver cette série dans S1;
Compter le nombre d'éléments de L2 par tranche de temps et sauver cette série dans S2;
Retourner le coefficient de corrélation linéaire Cor entre S1 et S2;

---

**Algorithm 1:** Algorithme de corrélation avec découpage régulier.

L'algorithme 2 présente la méthode de calcul de corrélation avec découpage en temps non-régulier.

La complexité globale de l'algorithme est en $O(n \times m)$, ou $n$ est le nombre d'éléments de la plus grande liste et $m$ le nombre d'éléments de l'autre liste.

Le filtrage d'une liste se fait en $O(n)$. Pour l'algorithme 1 le découpage de l'espace se fait en $O(1)$, et pour l'l'algorithme 2 en $O(n)$. Compter le nombre d'elements par liste est en $O(n)$ et $O(m)$. Le calcul du coefficient de corrélation est le produit des variances par la covariance, sa complexité dépends donc du calcul de la covariance ayant une complexité de $O(n \times m)$.

## 4 Correlation temporelle : implémentation dans FrameSoC

Nous avons implémenté les deux méthodes présentées dans la section précédente au sein de l'infrastructure FrameSoC. Nous exposons succinctement l'infrastructure FrameSoC dans une première partie. Nous présenterons ensuite les fonctionnalités de l'outil puis son interface.

### 4.1 Place de l'outil au sein de FrameSoC

FrameSoC est une infrastructure logicielle définissant des règles de programmation et gérant la cohérence entre plusieurs outils d'analyse de traces. La figure 8 représente l'architecture logicielle de FrameSoC. Les trois couches de l'architecture de FrameSoC concernent respectivement le stockage des traces (`MultiTrace DB`), l'accès aux données (`SoC-TRACE Library`) et les outils d'analyse (`GUI & Tools`). La couche de stockage défini un modèle de données générique pour les données des traces. L'implantation actuelle est faite au sein d'une base de données relationnelle

**Data**:
L1 : liste des éléments de type 1 ;
L2 : liste des éléments de type 2 ;
Delta : intervalle définissant la proximité pour un événement à problème ;
**Result**:
Cor : Coefficient de corrélation linéaire ;

**if** *filtrer la liste L1 ?* **then**
| filtrer la liste L1 selon la catégorie ;
**end**
**if** *filtrer la liste L2 ?* **then**
| filtrer la liste L2 selon la catégorie ;
**end**
Prendre comme espace de temps l'union des espaces temps des deux listes;
E1 = la plus petite série entre L1 et L2;
E2 = la plus grande série entre L1 et L2;
Utiliser E1 et Delta pour découper l'espace de temps en deux sous ensembles disjoints : tous les points à une distance inférieur à Delta des points de E1, tous les points à une distance supérieur à Delta des points de E1;
Compter le nombre d'éléments de L1 par tranche de temps et sauver cette série dans S1;
Compter le nombre d'éléments de L2 par tranche de temps et sauver cette série dans S2;
Retourner le coefficient de corrélation linéaire Cor entre S1 et S2;

**Algorithm 2:** Algorithme de corrélation avec histogramme non-régulier.

(SQLite [3] ) mais pourrait être basée sur un autre support de stockage comme par exemple un système de fichiers. L'accès aux données se fait via une bibliothèque fourni par FrameSoC qui propose les fonctions de base de manipulation de données comme la lecture, l'écriture ou le filtrage. Les outils FrameSoC se placent au sommet de cette architecture.

FrameSoC ainsi que les outils sont implémentés en Java et sont des plugins Eclipse. En termes de règles de programmation, les outils FrameSoC doivent étendre une classe abstraite qui sert de lien entre l'infrastructure et les outils. De cette manière, l'infrastructure connait les outils lancés et leur fournit des facilités d'accès à la base des traces et de visualisation. Plus de détails sur l'implémentation sont disponibles dans [12].

Pour intégrer nos traitements de détection d'anomalies et de calcul de corrélation temporelle entre événements, nous avons développé un outil FrameSoC spécifique. Notre outil d'analyse représente 8 classes Java pour un peu plus de 1000 lignes de code.

## 4.2 Interface fonctionnelle

L'outil que nous proposons permet d'obtenir de deux façons une corrélation entre deux types d'événements. La première de manière visuelle à l'aide de graphiques, la seconde en donnant le coefficient de corrélation linéaire.

La signature de la classe permettant l'analyse est la suivante :

3. www.sqlite.org

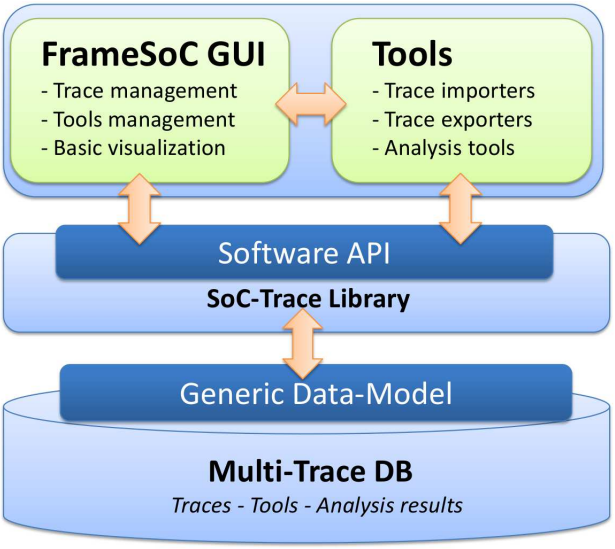



FIGURE 8 – Architecture logicielle de FrameSoC.

```
public StatsAnalysisManager(Trace t,
                            EventProducer ep1, EventType et1, boolean filter1,
                            EventProducer ep2, EventType et2, boolean filter2,
                            String analyse, Long delta);
```

Les données à fournir en entrée sont la trace que l'on souhaite analyser, ainsi que les deux séries d'événements que l'on veut corréler. Les derniers sont définis par leur type et leur producteur. Il est possible de demander à filtrer les éléments d'une série selon la catégorie des événements. Il faut aussi sélectionner laquelle des deux méthodes présentées en section 3 (nuage de points ou histogramme ) il faut utiliser pour la représentation.

Après le calcul, l'outil affichera en résultat la répartition des événements selon le temps sur un premier graphique. Sur un second, l'histogramme de densité de ces événements selon les tranches de temps dans lequel l'espace à été découpé. Un résultat numérique est aussi affiché et correspond au coefficient de corrélation linéaire présenté en section 2.

## 4.3 Interface graphique

La Figure 9 montre l'interface graphique de l'outil au démarrage de celui-ci.

Il faut dans un premier temps sélectionner la trace que l'on veux analyser. L'outil propose de choisir entre toutes les traces présentent dans la base de donnée de FrameSoC. L'outil propose ensuite les types d'événements correspondant à la trace sélectionnée. Il est possible de choisir de filtrer ou non l'une ou les deux séries. Un bouton radio permet de définir l'une ou l'autre méthode de calcul. Il est aussi nécessaire de rentrer une valeur dans la case "delta", celle-ci sera utilisée lors du calcul avec l'histogramme non régulier pour définir les éléments "aux alentours" des points de la plus petite série. Le bouton "GO" permet de lancer l'analyse avec en paramètre les données sélectionnées. La figure 10 montre un exemple d'entrées d'analyse.

Une fois les calculs effectués, l'outil affiche deux graphiques comme montré sur les Figures 11 et 12. Le graphique de gauche représente les événements pour chaque série. Un point représente

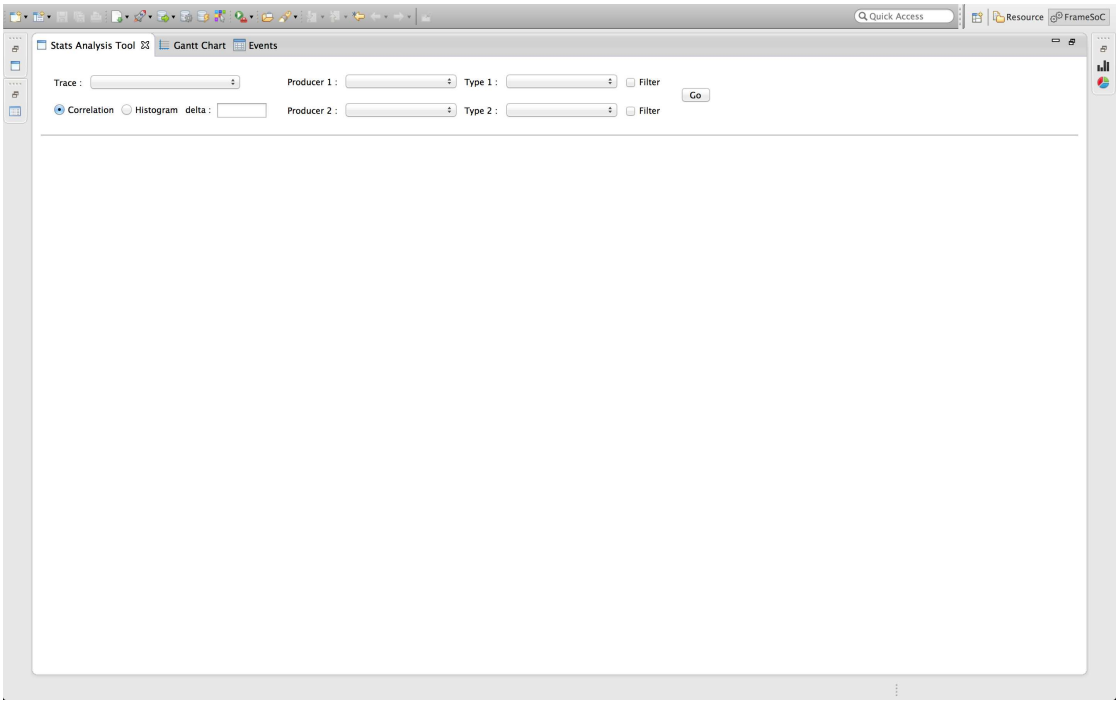



FIGURE 9 – Vue d'ensemble de l'outil au démarrage.

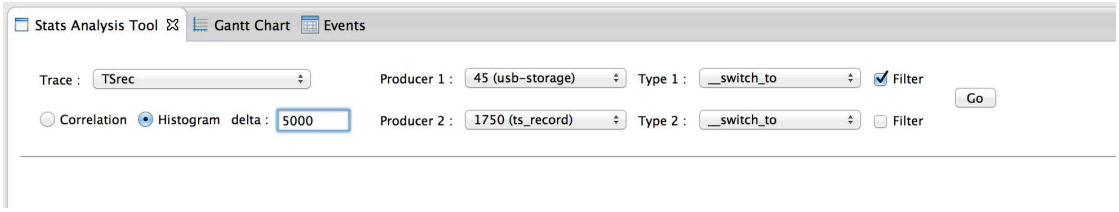



FIGURE 10 – Exemple de paramètres d'entrée.

un événement. En abscisse on à l'estampille de l'événement et en ordonnée le numéro de l'événement. Cette représentation permet de palier le manque de visibilité si plusieurs événements sont intervenues de manière trop proche dans le temps. On retrouve le même graphique que dans la Figure 5 qui représente le cas *TSrecord*. Le graphique de droite représente selon la méthode choisie soit le nuage de points de l'histogramme régulier (Figure 11), soit l'histogramme des densités des événements pour la méthode avec histogramme non régulier (Figure 12). Dans cet exemple on retrouve dans la figure de l'histogramme les intervalles déterminés comme dans la figure 7. En bleu est représenté le nombre d'éléments de la série dans les intervalles aux alentours des anomalies, en rouge le nombre d'elements entre ces intervalles.

## 4.4 Evaluation

Nous avons évalué notre outil en utilisant le cas TSrecord présenté en section 3.1.2. Nous avons utilisé Eclipse en version 4.3 et java 1.7. La machine que nous avons utilisé est un MacBook Pro avec un CPU Intel i7 quadri-core 2.6 GHz, 16Go de RAM, et utilisant MacOS 10.9.

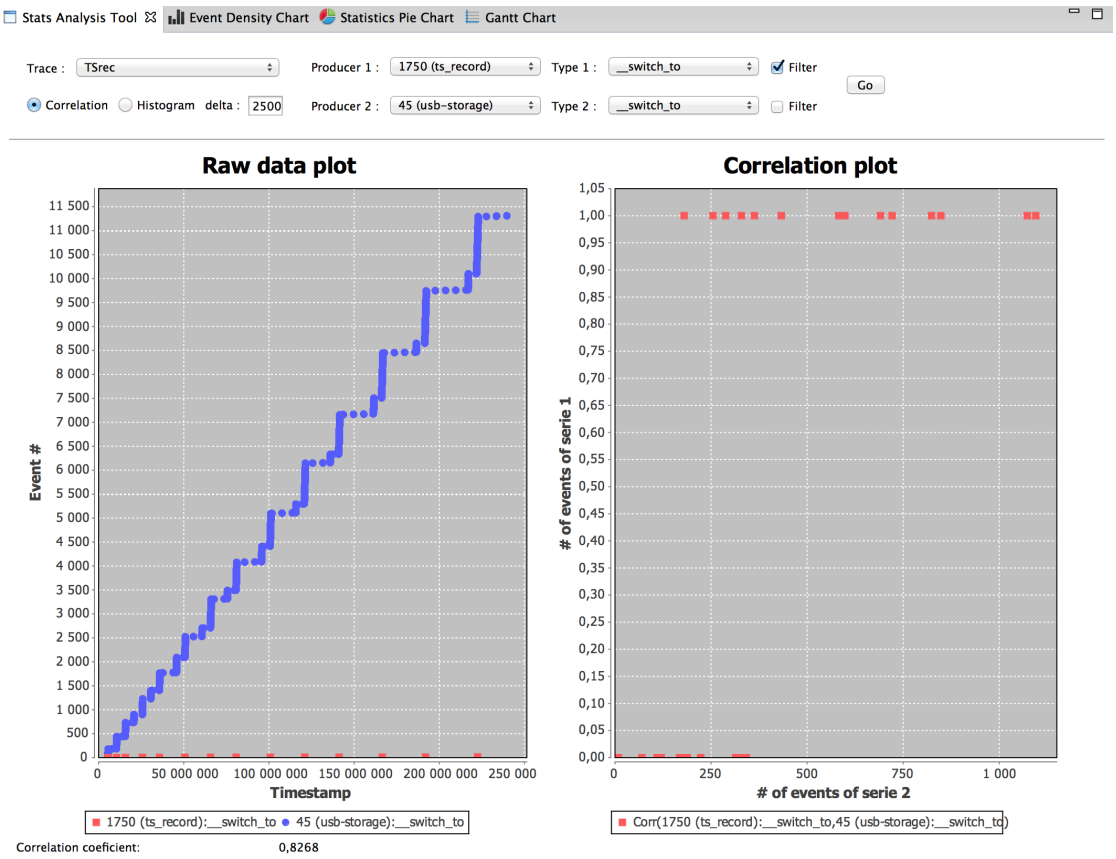



FIGURE 11 – Exemple de résultat par corrélation avec découpage régulier.

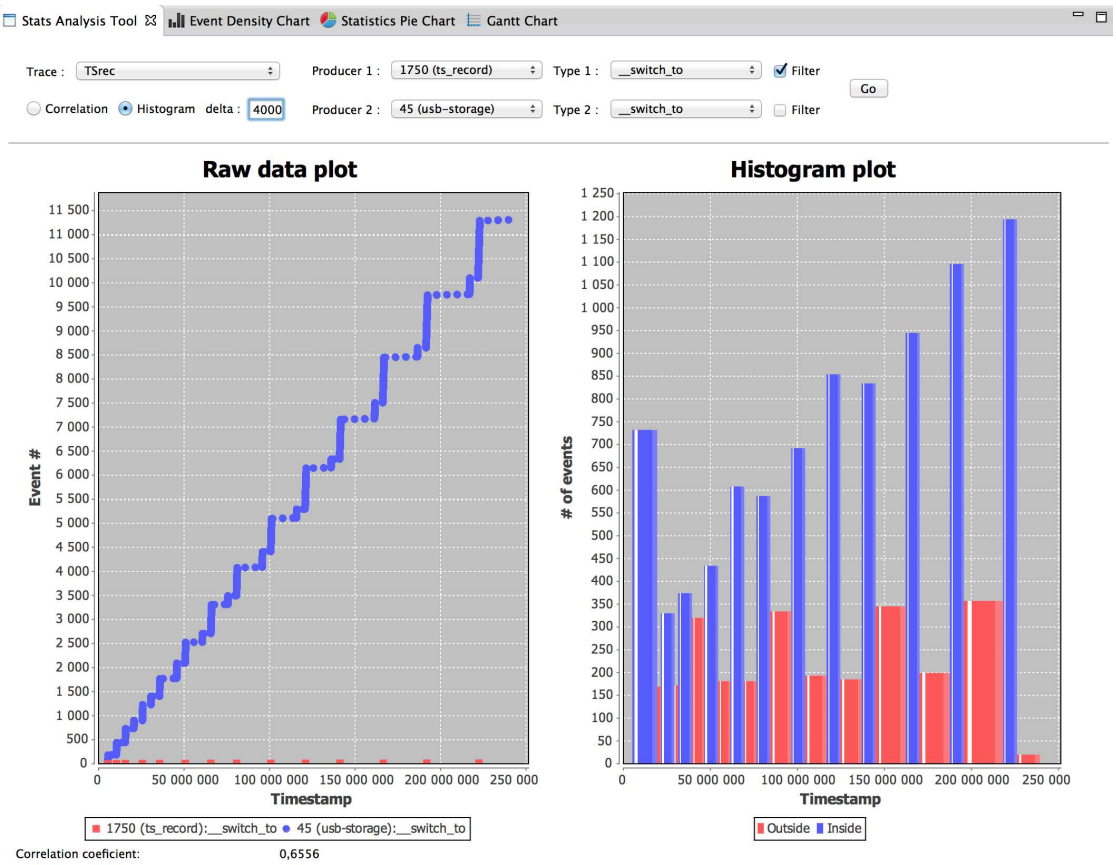



FIGURE 12 – Exemple de résultat par corrélation avec découpage non régulier.

La trace considérée contient au total 814 289 événements. Les deux séries que nous comparons contiennent 23 408 et 11 312 événements. Il faut environs 9,2 secondes pour récupérer ces deux séries d'événements. Nous filtrons la première série, cela prends environ 3 secondes. Le calcul du coefficient de correlation prends environ 300 ms.

Sur cet exemple, le temps total de l'analyse est d'environ 12 à 14 secondes selon la méthode utilisée.

Dans les deux cas la mesure de correlation obtenue nous indique une forte correlation entre ces deux séries. Dans le cas de l'histogramme régulier, la mesure est unique car elle ne dépends que des deux séries données en entrée. Pour l'histogramme non-régulier, l'utilisation d'un delta variable définit par l'utilisateur nécessite plusieurs calculs afin de trouver une valeur faisant ressortir cette correlation.

# 5 Méthodes existantes d'analyse de données

Plusieurs domaines scientifiques traitent de l'analyse de données. Nous nous sommes intéressés aux séries temporelles, au traitement du signal ainsi qu'au méthodes de clustering.

**Séries temporelles** Les séries temporelles sont utilisées pour représenter l'évolution d'une quantité au cours du temps. Elles sont notamment utilisés en économie car elle permettent de comprendre l'évolution passée et de prédire l'evolution future de ces quantités. L'étude des séries temporelles permet typiquement de repérer des tendances dans l'évolution des quantités (croissance, décroissance, linéaire, quadratique) et saisonnières (périodicité)[2] .

Les méthodes d'analyse de séries temporelles ne s'appliquent pas aux traces d'exécution. En effet, une trace ne représente pas l'évolution d'une quantité dans le temps mais est une suite d'événements estampillés qui ne sont pas forcément reliés.

**Traitement du signal** Le traitement du signal est la discipline qui consiste à appliquer à un signal, c'est à dire une série de valeurs numériques, des opérations afin d'analyser celui-ci[6, 15, 5]. Dans ce cadre, un signal est typiquement représenté par un échantillonnage périodique. Un traitement largement utilisé est la transformée de Fourier qui permet de détecter la périodicité de phénomènes.

Les traces d'exécutions peuvent être vues comme des signaux discrets avec plusieurs canaux, chaque canal représentant un type d'événement. Néanmoins, les traitements du signal ne peuvent être appliqués puisque les événements d'une trace ne représentent pas un échantillonnage régulier dans le temps.

**Clustering** Le clustering consiste à regrouper des éléments d'un ensemble en utilisant des critères de similarité [11]. Le clustering est utilisé, par exemple, dans la reconnaissance d'images ou dans le domaine du data mining. Dans le cas des traces d'exécution, nous pouvons raisonner en termes d'ensemble d'événements où la notion de similarité qui nous intéresse est leur proximité temporelle.

Il est envisageable d'utiliser des algorithmes comme DBSCAN [7] afin de détecter la proximité temporelle d'événements au sein d'une trace. Les clusters identifiés contiendrait les événements se produisant en même temps ou à des intervalles de temps très courtes. Pour ce faire, il faudrait décider d'un seuil caractérisant la distance : jusqu'à quelle distance de temps peut-on considérer que deux événements sont proches (appartiennent au même cluster) et à partir de quelle distance

peut-on considérer qu'ils sont éloignés ? Ce paramètre devrait être décidé en fonction du contexte général de la trace.

Dans le cas de détection d'anomalies, le clusters devraient être construits autour de pivots qui seraient les événements anormaux de la trace. Or, les algorithmes classiques de clustering n'intègrent pas des contraintes de ce type-là.

# 6 Conclusion

Dans ce rapport nous avons proposé une méthode d'analyse de traces consistant à appliquer des traitements statistiques simples afin de détecter des anomalies, d'une part, et corréler ces anomalies à d'autres phénomènes d'exécution, d'autre part. Nous avons implémenté un outil au sein de l'infrastructure FrameSoC qui caractérise la corrélation temporelle entre deux séries d'événements en utilisant la visualisation et le calcul d'une mesure de corrélation. Nous avons appliqué notre méthode avec succès à deux scénarios de décodage multimédia fournis par STMicroelectronics.

Dans l'état actuel du prototype, l'utilisateur doit choisir manuellement les types d'événements à considérer afin de chercher des anomalies. A court terme, nous pouvons envisager une automatisation de la procédure afin de considérer toutes les séries d'événements.

La détection d'anomalies est principalement basée sur la notion de périodicité et de durée d'exécution. Il serait intéressant d'approfondir cet aspect en considérant d'autres critères comme par exemple, les événements rares ou la concentration d'occurrences d'événements sur une partie de la trace.

A long terme, notre travail se place dans un contexte où il n'existe pas de traitements reconnus d'analyse de traces et où l'analyse se fait de manière ad-hoc au cas par cas. Nous nous intéressons à la définition de briques génériques d'analyse de traces afin de définir une bibliothèque ouverte et réutilisable. Dans un contexte où les traces d'exécution sont de plus en plus utilisées pour comprendre et optimiser le comportement de systèmes complexes et où leur volume ne cesse de croitre, disposer d'une base de traitements simplifierait et rendrait plus rapide leur analyse. Une telle bibliothèque permettrait la définition de chaînes d'analyse qui à terme pourraient être automatisées.

# Table des matières

# Références

[1] Correlation (in statistics). a.v. prokhorov (originator), encyclopedia of mathematics. http://www.encyclopediaofmath.org/index.php?title=Correlation_(in_statistics)&oldid=11629.

[2] Nist/sematech e-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4.htm.

[3] Projet SoC-Trace. http://tinyurl.com/minalogic-soc-trace/.

[4] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[5] JF Ames. *A Student's Guide to Fourier Transforms (3rd edition)*. 2011.

[6] Proakis Dimitris. *Digital Signal Processing (3rd edition)*. 1996.

[7] Martin Ester, Hans-peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[8] LTTng Viewers. https://lttng.org/viewers.

[9] Vania Marangozova-Martin and Generoso Pagano. Gestion de traces d'exécution pour le systèmes embarqués : contenu et stockage. Research Report RR-LIG-046, LIG, Grenoble, France, 2013.

[10] Alexis Martin, Generoso Pagano, Jérôme Correnoz, and Vania Marangozova-Martin. Analyse de systèmes embarqués par structuration de traces d'exécution. In *ComPAS'2014*, Neuchâtel, Suisse, April 2014.

[11] M N Murty, A K Jain, and P J Flynn. Data Clustering : A Review. *ACM Computing Surveys*, 31(3), 1999.

[12] Generoso Pagano and Vania Marangozova-Martin. SoC-Trace Infrastructure. Technical Report RT-427, 2012.

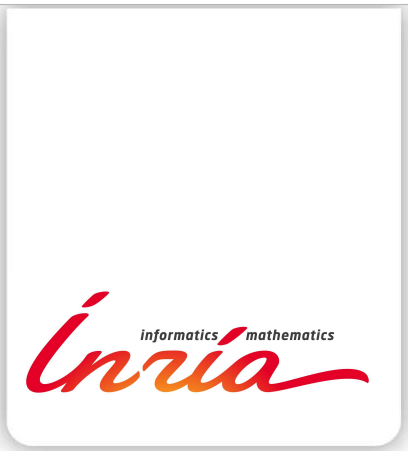
[13] K Pearson. Contributions to the Mathematical Theory of Evolution. II. Skew Variation in Homogeneous Material. 1896.

[14] Sheldon Ross. *Introduction to Probability and Statistics for Engineers and Scientists*. Academic Press, 2009.

[15] Steven W Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. 1999.

[16] B De Oliveira Stein. *Visualisation Interactive et Extensible de Programmes Parallèles à Base de Processus Légers*. PhD thesis, 1999.

[17] STMicroelectronics. STWorkbench for STLinux & OS21. http://stlinux.com/devel/stwbwebcasts.




**RESEARCH CENTRE**
**GRENOBLE – RHÔNE-ALPES**

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803