

Monitoring et détection d'anomalie par apprentissage dans les infrastructures virtualisées

Carla Sauvanaud

▶ **To cite this version:**

Carla Sauvanaud. Monitoring et détection d'anomalie par apprentissage dans les infrastructures virtualisées. Autre [cs.OH]. INSA de Toulouse, 2016. Français. NNT : 2016ISAT0043 . tel-01445648v2

HAL Id: tel-01445648
https://theses.hal.science/tel-01445648v2

Submitted on 22 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :
l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *13/12/2016* **par :**

CARLA SAUVANAUD

MONITORING ET DÉTECTION D'ANOMALIE PAR APPRENTISSAGE DANS DES INFRASTRUCTURES VIRTUALISÉES

JURY

SARA BOUCHENAK	Professeur des Universités, INSA de Lyon	Rapporteur
PIERRE SENS	Professeur des Universités, LIP6	Rapporteur
KAHINA LAZRI	Ingénieur de recherche, Orange Labs	Examinateur
ERNESTO EXPOSITO	Professeur des Universités, LIUPPA	Examinateur
MOHAMED KAÂNICHE	Directeur de recherche, CNRS	Directeur de thèse
KARAMA KANOUN	Directeur de recherche, CNRS	Directeur de thèse

École doctorale et spécialité :
EDSYS : Informatique 4200018
Unité de Recherche :
Laboratoire d'analyse et d'architecture des systèmes
Directeur(s) de Thèse :
Mohamed Kaâniche et *Karama Kanoun*
Rapporteurs :
Sara Bouchenak et *Pierre Sens*

Everyone has a plan 'till they get punched in the face.
Mike Tyson

Remerciements

Les travaux présentés dans ce manuscrit ont été effectués au sein du Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS). Je remercie Messieurs Jean Arlat et Liviu Nicu, qui ont successivement assuré la direction du LAAS-CNRS depuis mon entrée, pour m'avoir accueilli au sein de ce laboratoire. Je remercie également Monsieur Mohamed Kaâniche et Madame Karama Kanoun, directeurs de recherche CNRS, responsables successifs du département Informatique critique de l'équipe Tolérance aux fautes et Sûreté de Fonctionnement informatique (TSF), pour m'avoir permis de réaliser ces travaux dans cette équipe qui offre dans un cadre propice au travail et aux échanges.

Je remercie de plus Monsieur Mohamed Kaâniche et Madame Karama Kanoun en tant que directeurs de mes travaux de thèse, pour m'avoir guidé durant ces trois années de thèse. Ces travaux n'auraient pas pu aboutir sans leur concours, leurs précieux conseils, questionnements pertinents et leur disponibilité malgré leurs emplois du temps très chargés. J'adresse également mes sincères remerciements aux membres du jury qui ont accepté de juger mon travail :

— Sara Bouchenak, Professeur des Universités, INSA de Lyon, Rapporteur ;
— Pierre Sens, Professeur des Universités, LIP6, Rapporteur ;
— Kahina Lazri, Ingénieur de recherche, Orange Labs, Examinateur ;
— Ernesto Exposito, Professeur des Universités, LIUPPA, Examinateur ;
— Mohamed Kâaniche, Directeur de recherche, CNRS, Directeur de thèse ;
— Karama Kanoun, Directeur de recherche, CNRS, Directeur de thèse.

Je suis reconnaissante de l'intérêt qu'ils ont porté à mes travaux.

Mes remerciements s'adressent aussi à toutes les personnes qui se sont penchées avec intérêt sur mes travaux notamment les collaborateurs du projet d'investissements d'avenir Secure Virtual Cloud dans lequel s'inscrivent mes travaux de thèse, et tout particulièrement Kahina Lazri de Orange Labs qui a proposé le deuxième cas d'étude de mes travaux et avec qui j'ai acquis des compétences sur les fonctions réseau virtualisées. Je remercie également Kalou Cabrera, Juliette Dromard pour avoir relu mon manuscrit de thèse ainsi qu'Ulrich Aivodji et Thierry Sotiropoulos pour m'avoir écouté présenter mes travaux à plusieurs reprises. Mes pensées vont de plus vers ma famille que je remercie pour leur soutien inconditionnel et tout particulièrement à mon petit frère Hugo.

Je remercie l'ensemble du personnel du département Génie Électrique et Informatique de l'INSA de Toulouse pour m'avoir intégré dans leur équipe d'enseignement et donné des responsabilités de manière à ce que je puisse acquérir les connaissances nécessaires à la poursuite d'une carrière dans l'enseignement. Je remercie particulièrement Christophe Chassot, Vincent Nicomette, Éric Alata, Daniela Dragomirescu, Pierre-Emmanuel Hladik, Ernesto Exposito, Nawal Guermouche, Slim Abdellatif et Thierry Gaffier.

Je remercie enfin toutes les personnes de l'équipe TSF et notamment les doctorants qui ont participé à faire de mes travaux de thèse une belle aventure humaine et

scientifique et en pariculier Miruna Stoicescu, Anthony Dessiatnikoff, Miguel Nuñez Del Prado, Benoît Morgan, William Excoffon et Thierry Sotiropoulos qui ont chacun participé à des moments différents et de manière complémentaire à amener une fabuleuse atmosphère dans le bureau que nous partagions à TSF.

Table des matières

1 Contexte et concepts fondamentaux **9**
1.1 Le cloud computing 9
1.1.1 Définition et caractéristiques 10
1.1.2 La virtualisation 11
1.2 Problématique de sûreté de fonctionnement dans les clouds 12
1.2.1 La sûreté de fonctionnement 12
1.2.1.1 Attributs de la sûreté de fonctionnement 13
1.2.1.2 Entraves à la sûreté de fonctionnement 14
1.2.1.3 Moyens pour la sûreté de fonctionnement 14
1.2.2 Détection d'anomalie dans les services clouds 15
1.2.2.1 Caractéristiques de la détection d'anomalies dans les services cloud 15
1.2.2.2 Critères pour la sélection d'une technique de détection d'anomalies 17
1.3 Conclusion 18

2 Détection d'anomalie par apprentissage - État de l'art **19**
2.1 Détection d'anomalie 20
2.1.1 Différents types de données 20
2.1.2 Techniques basées sur les statistiques, les théories des probabilités et de l'information 22
2.1.3 Techniques de détection basées sur l'apprentissage automatique 23
2.1.3.1 Détection par apprentissage supervisé 24
2.1.3.2 Détection par apprentissage non supervisé 26
2.1.3.3 Discussion sur les besoins de détection 28
2.2 Évaluation expérimentale de performances de détection 29
2.2.1 Injection de fautes 30
2.2.2 Métriques d'évaluation 31
2.3 Conclusion 34

3 Présentation générale de la stratégie de détection **37**
3.1 Cadre conceptuel 37
3.1.1 Contexte 38
3.1.2 Hypothèse sur la manifestation d'une anomalie 38
3.2 Entité de monitoring 39

3.2.1 Différentes sources de monitoring . . . 39
3.2.2 Groupement des compteurs de performance . . . 40
3.2.2.1 Groupement par-VM . . . 40
3.2.2.2 Groupement par-ressource . . . 41
3.2.2.3 Discussion sur le diagnostic . . . 41
3.3 Entité de détection . . . 42
3.3.1 Niveau de détection . . . 42
3.3.2 Type de détection . . . 44
3.3.3 Détection par apprentissage supervisé . . . 44
3.3.3.1 Fonctionnement en deux phases . . . 44
3.3.3.2 Groupement des données . . . 46
3.3.4 Détection DESC . . . 46
3.3.4.1 Analyse des données de monitoring et hypothèses sur les clusters . . . 47
3.3.4.2 Groupement des données . . . 48
3.3.4.3 Phase de caractérisation de comportement . . . 49
3.3.4.4 Test et discussion à propos de la caractérisation de comportement . . . 53
3.3.4.5 Phase d'analyse de comportement . . . 54
3.3.4.6 Présentation de la technique en pseudo-code . . . 55
3.3.4.7 Choix des paramètres de clustering . . . 56
3.4 Conclusion . . . 57

4 Mise en œuvre 59
4.1 Plateforme d'expérimentation . . . 60
4.2 Module de monitoring . . . 60
4.2.1 Outils de monitoring . . . 61
4.2.2 Compteurs de performance . . . 61
4.2.3 Période de monitoring . . . 62
4.3 Module de charge de travail . . . 62
4.4 Module d'injection de fautes . . . 62
4.4.1 Erreurs injectées . . . 63
4.4.1.1 Causes émulées et moyens utilisés . . . 63
4.4.1.2 Réflexion sur le diagnostic . . . 65
4.4.2 Campagne d'injection . . . 66
4.4.2.1 Description . . . 66
4.4.2.2 Intensité d'injection . . . 67
4.4.2.3 Durée d'injection . . . 68
4.4.2.4 Temps de pause . . . 68
4.4.3 Autres types de perturbations . . . 68
4.5 Module de détection . . . 70
4.5.1 Mise en œuvre des différents types de détection . . . 70
4.5.2 Détection par apprentissage supervisé . . . 71
4.5.3 Détection DESC . . . 72

4.6 Évaluation 74
4.6.1 Exploitation des données 74
4.6.2 Évaluation des performances 75
4.6.3 Analyse de compteurs de performance 75
4.7 Conclusion 76

5 Évaluation du cas d'étude SGBD : MongoDB **77**
5.1 Présentation 78
5.1.1 Description et déploiement 78
5.1.2 Charge de travail 79
5.1.3 Jeux de données 79
5.1.3.1 Description 79
5.1.3.2 Utilisation 80
5.2 Évaluation de la détection par apprentissage supervisé 80
5.2.1 Comparaison d'algorithmes : performances de détection . . . 80
5.2.2 Comparaison d'algorithmes : temps d'exécution 81
5.2.3 Comparaison d'algorithmes : durée de validité du jeu d'entrainement 82
5.2.4 Détection d'erreurs 82
5.3 Évaluation de la détection DESC hybride 83
5.3.1 Détection d'erreurs 83
5.3.2 Durée de validité du jeu d'entrainement 85
5.4 Analyse des compteurs de performance utiles à la détection 88
5.5 Conclusion 89

6 Évaluation du cas d'étude VNF : Clearwater **93**
6.1 Présentation 94
6.1.1 Description et déploiement 94
6.1.2 Charge de travail 95
6.1.3 Jeux de données 95
6.1.3.1 Description 95
6.1.3.2 Utilisation 97
6.2 Évaluation de la détection par apprentissage supervisé 97
6.2.1 Détection d'erreurs 97
6.2.1.1 Sensibilité à la distribution des données 97
6.2.1.2 Détection avec diagnostic de types d'erreur 98
6.2.1.3 Détection avec diagnostic de la VM à l'origine d'erreur 99
6.2.1.4 Sensibilité aux intensités d'injection dans le jeu de données d'entrainement 100
6.2.1.5 Sensibilité à la taille du jeu de données 101
6.2.1.6 Sensibilité à la durée d'injection 102
6.2.1.7 Durée de validité du jeu d'entrainement 104
6.2.2 Détection de symptômes préliminaires à une violation de service 107

6.2.2.1 Détection avec diagnostic de la VM à l'origine d'une violation de service . . . 108
6.2.2.2 Durée de validité du jeu d'entrainement . . . 108
6.2.3 Détection de violations de service . . . 109
6.2.3.1 Détection avec diagnostic du type d'erreur à l'origine de la violation de service . . . 110
6.2.3.2 Détection avec diagnostic de la VM à l'origine d'une violation de service . . . 111
6.2.3.3 Comparaison avec le monitoring SE . . . 111
6.2.3.4 Sensibilité à la durée d'injection . . . 112
6.2.3.5 Durée de validité du jeu d'entrainement . . . 113
6.3 Évaluation de la détection DESC hybride . . . 114
6.3.1 Détection d'erreurs . . . 115
6.3.2 Détection de violations de service . . . 117
6.3.3 Durée de validité du jeu d'entrainement . . . 117
6.4 Analyse des compteurs de performance utiles à la détection . . . 119
6.5 Conclusion . . . 128

7 Conclusions et perspectives 133

A Monitoring 137
A.1 Compteurs de performance en monitoring SE . . . 137
A.1.1 Compteurs utilisés avec un apprentissage supervisé . . . 137
A.1.2 Compteurs utilisés avec la détection DESC . . . 139
A.1.2.1 Sous-flux CPU . . . 139
A.1.2.2 Sous-flux mémoire . . . 139
A.1.2.3 Sous-flux disque . . . 139
A.1.2.4 Sous-flux réseau . . . 139
A.2 Compteurs de performance en monitoring hyperviseur . . . 140

B Appel SIPp pour le cas d'étude Clearwater 143

Bibliographie 151

Table des figures

1.1.1 Principaux types d'hyperviseurs. 12
1.2.1 Arbre de la sûreté de fonctionnement [Avizienis 2004] 13

2.2.1 Courbes ROC et PR pour une classification parfaite (en rouge) et une classification aléatoire (en bleu). 33

3.2.1 Sources de monitoring. 40
3.2.2 Groupement des compteurs par-VM. 41
3.2.3 Groupement des compteurs par-ressource. 42
3.3.1 Représentation de PE pour un service correct, des symptômes préliminaires à une violation de service et d'une violation de service. 43
3.3.2 Deux phases d'exécution de la technique de détection par apprentissage supervisé. 45
3.3.3 Technique de caractérisation de comportement. 50
3.3.4 Attribut DtR représenté pour un espace de 2 dimensions. 52
3.3.5 Attribut CS représenté pour un espace de 2 dimensions. 52
3.3.6 Attribut DtR pour le sous-flux disque, en fonction du temps durant des tests d'injection d'anomalie. 53
3.3.7 Diminution du poids d'un cluster sans nouvelle observation en fonction de λ. 57

4.1.1 Plateforme d'expérimentation. 60
4.4.1 Diagramme illustrant une campagne d'injection. 67

5.1.1 Déploiement MongoDB. 79
5.2.1 Détection d'erreurs binaire avec plusieurs algorithmes à apprentissage supervisé. 81
5.2.2 Détection d'erreurs binaire avec plusieurs algorithmes à apprentissage supervisé et pour un entrainement sur $\mathcal{Y}$ et une prédiction sur $\mathcal{Z}$. 83
5.2.3 Détection d'erreurs selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{Y}$ en *Detect_Sup*. 84
5.2.4 Détection d'erreurs selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{Z}$ en *Detect_Sup*. 85
5.3.1 Détection d'erreurs selon le type d'erreur locale à la VM observée en *Detect_DESC_hybride* par sous-flux. 86

5.3.2 Détection d'erreurs selon le type d'erreur locale à la VM observée, en *Detect_DESC_hybride* agrégée. 86
5.3.3 Détection d'erreurs selon le type d'erreur locale à la VM observée, avec un apprentissage sur $\mathcal{Y}$ et une prédiction sur $\mathcal{Z}$, en *Detect_DESC_hybride* agrégée. 87
5.3.4 Détection binaire d'erreur, avec un apprentissage sur $\mathcal{Y}$ et une prédiction sur $\mathcal{Z}$, en *Detect_DESC_hybride* agrégée. 87

6.1.1 Déploiement Clearwater. 95
6.2.1 Détection binaire d'erreurs avec plusieurs distributions en *Detect_Sup_RF*. 98
6.2.2 Détection d'erreurs selon le type d'erreur locale à la VM observée en *Detect_Sup_RF*. 99
6.2.3 Détection d'erreurs selon la VM à l'origine de l'erreur en *Detect_Sup_RF*. 100
6.2.4 Détection d'erreurs selon le type d'erreur locale à la VM observée avec un entrainement sur deux intensités de chaque erreur représentée dans $\mathcal{A}$ et une prédiction sur sept intensités de chaque erreur représentée dans $\mathcal{A}$ en *Detect_Sup_RF*. 101
6.2.5 Détection d'erreurs selon la VM à l'origine de l'injection avec le jeu de données $\mathcal{A}$ réduit de 75% en *Detect_Sup_RF*. 102
6.2.6 Détection d'erreurs selon le type d'erreur locale à la VM observée et 4 min d'injection en *Detect_Sup_RF*. 103
6.2.7 Détection d'erreurs selon la VM à l'origine de l'erreur et 4 min d'injection en *Detect_Sup_RF*. 103
6.2.8 Détection binaire d'erreurs avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*. 105
6.2.9 Détection d'erreurs selon le type d'erreur locale à la VM observée avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*. 105
6.2.10 Détection d'erreurs selon le type d'erreur locale à la VM observée avec un apprentissage sur $\mathcal{B}$ et une prédiction sur $\mathcal{D}$ (décalage de 1 semaine entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*. 106
6.2.11 Détection d'erreurs selon le type d'erreur locale à la VM observée en monitoring SE avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*. 107
6.2.12 Détection de symptômes préliminaires à des violations de service selon la VM à l'origine de la violation en *Detect_Sup_RF*. . . . 108

6.2.13 Détection binaire de symptômes préliminaires à des violations de service avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*. 109
6.2.14 Détection binaire de symptômes préliminaires à des violations de service avec un apprentissage sur $\mathcal{B}$ et une prédiction sur $\mathcal{D}$ (décalage de 1 semaine entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*. 110
6.2.15 Détection de violations de service selon le type d'erreur locale à la VM observée en *Detect_Sup_RF*. 111
6.2.16 Détection de violations de service selon la VM à l'origine de la violation en *Detect_Sup_RF*. 112
6.2.17 Détection de violations de service selon la VM à l'origine de la violation en monitoring SE en *Detect_Sup_RF*. 113
6.2.18 Détection de violations de service selon le type d'erreur locale à la VM observée et 4 min d'injection en *Detect_Sup_RF*. 114
6.2.19 Détection de violations de service selon la VM à l'origine de la violation de service avec un apprentissage sur $\mathcal{A}$ et une détection sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*. 115
6.2.20 Détection de violations de service selon le type d'erreur locale à la VM observée en monitoring SE avec un apprentissage sur $\mathcal{A}$ et une détection sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*. . 116
6.3.1 Détection d'erreurs selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{A}$ en *Detect_DESC_hybride*. 117
6.3.2 Détection d'erreurs selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{B}$ en *Detect_DESC_hybride*. 118
6.3.3 Détection de violations de service selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{B}$ en *Detect_DESC_hybride*.118
6.3.4 Détection binaire d'erreurs avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_DESC_hybride*. . . . 119
6.3.5 Détection binaire de violation de service avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_DESC_hybride*. 120

Liste des tableaux

2.1 Matrice de confusion. 32
2.2 Définition des mesures de performance. 32

4.1 Intensité des injections. 67
4.2 Exemple d'un jeu d'observations dans le premier cas exprimé en pourcentages et dans le deuxième cas avec les métriques centrées réduites. 72

5.1 Temps d'entrainement et de prédiction (ms). 82
5.2 Compteurs les plus utiles en monitoring hyperviseur pour une détection multi-classe selon le type d'erreur locale à la VM observée . . . 91

6.1 Compteurs les plus utiles en monitoring hyperviseur en détection binaire d'erreurs et de violation de service. 123
6.2 Compteurs les plus utiles en monitoring hyperviseur en détection avec diagnostic selon la VM à l'origine d'erreur et de violation de service. 124
6.3 Compteurs les plus utiles en monitoring hyperviseur en détection multi-classe selon le type d'erreur locale à la VM observée et de violation de service. 125
6.4 Compteurs les plus utiles en monitoring hyperviseur en détection binaire et multi-classe de symptômes préliminaires de violations de service. 126
6.5 Compteurs les plus utiles en monitoring hyperviseur en détection multi-classe selon le type d'erreur locale à la VM observée 128
6.6 Compteurs les plus utiles en monitoring SE et détection multi-classe selon le type d'erreur locale à la VM observée 130

Introduction générale

Contexte général

De nos jours, l'essor des technologies de virtualisation ainsi que de l'Internet a contribué à l'émergence de nouveaux paradigmes informatiques tels que l'*informatique dans les nuages* ou le *cloud computing* en anglais. Le cloud computing est un modèle de délivrance par un accès réseau ubiquitaire, pratique et à la demande, d'un ensemble de ressources informatiques partagées et configurables. Ces ressources sont détenues par un fournisseur de service cloud et peuvent être aussi variées que des applications, des plateformes de développement ou bien des infrastructures. Ces services sont respectivement appelés SaaS pour Software as a Service en anglais, PaaS pour Platform as a Service, et IaaS pour Infrastructure as a Service.

Comme tout système informatique, le cloud n'est pas exempt des problématiques de sûreté de fonctionnement. Dans nos travaux, nous nous intéressons à celles liées à la disponibilité et la fiabilité. Assurer ces deux propriétés pour des services à la fois différents, en allocation dynamique et pour des utilisateurs aux demandes hétérogènes représente un défi pour les fournisseurs, notamment par leur engagement de service à la demande. Ce défi est d'autant plus important que les utilisateurs demandent à ce que les services rendus soient au moins aussi sûrs de fonctionnement que ceux d'applications traditionnelles.

Nos travaux traitent particulièrement de la détection d'anomalies dans les services cloud de type SaaS et PaaS. Nous visons à détecter une anomalie dans un système tout en fournissant des indicateurs permettant d'identifier la cause de cette anomalie. Ces indicateurs quant à eux permettent de mettre en place des mécanismes de recouvrement.

Nous définissons que la détection d'anomalies dans les clouds se réalise sous quatre critères principaux à prendre en compte pour la sélection d'une technique de détection.

Premièrement, la détection d'anomalies doit prendre en compte les changements de charge de travail et les reconfigurations possibles des services cloud. Deuxièmement, la complexité des services fait que la première caractéristique énoncée ne peut être assurée qu'avec des techniques de traitement automatique des données. Alors que dans des systèmes de contrôle/commande dont la ou les applications sont connues à l'avance il est possible de connaître les caractéristiques de l'application

a priori, il est également possible de déduire des ensembles d'états de fonctionnement et de reconnaitre des situations ou comportements anormaux. Toutefois, plus un système est grand avec des tâches variées, voire inconnues a priori, plus il est difficile de dissocier un état de comportement normal d'un état de comportement anormal. Pour cette raison, il est nécessaire dans un cloud de faire appel à des techniques automatiques définissant des comportements (normaux ou anormaux) d'un système. Troisièmement, la technique de détection doit être applicable à tout type de service cloud afin de minimiser le temps de configuration et de déploiement. Quatrièmement, la détection doit de plus se faire en ligne, en continu et soit en amont soit très rapidement après l'occurrence d'une anomalie, afin que l'utilisateur du service ne subisse pas de défaillance importante.

Présentation de nos travaux

Les travaux de ce manuscrit de thèse définissent notre stratégie de détection d'anomalies dans les services cloud, répondant aux critères principaux que nous avons définis.

Cette stratégie repose sur des techniques d'apprentissage automatique exploitant les données de performance système collectées en ligne depuis les services clouds déployés par un fournisseur. Cette stratégie peut être appliquée indépendamment du service étudié car elle repose sur l'analyse de données de performance système accessibles à partir de tout système informatique. Nous montrons dans ce manuscrit que son implémentation est de plus très peu dépendante de ce dernier. Elle repose également sur une technique d'injection de fautes permettant d'introduire des anomalies dans un système cible afin d'entrainer les modèles d'apprentissage automatique à discriminer les comportements normaux des anomalies.

Notre stratégie repose donc sur deux entités dont l'action peut être menée de différentes manières.

L'entité de collecte de données de performance système travaille avec deux sources différentes. L'une collecte des données à partir des systèmes d'exploitation hôte du service considéré et l'autre collecte des données grâce aux hyperviseurs déployés qui hébergent ce service dans le cloud.

L'entité de détection fonctionne avec des niveaux de détection d'anomalies différents, à savoir la détection d'erreurs, de symptômes préliminaires à une violation de service, et de violations de service. Plusieurs types de détection peuvent également être mis en œuvre en fonction d'un diagnotic réalisé quant à l'origine supposée d'une anomalie en cours. Enfin, deux techniques d'apprentissage automatique sont proposées afin de construire des modèles de détection d'anomalies selon les différents niveaux et le diagnostic voulu. La première repose sur un algorithme à apprentissage supervisé. La seconde repose sur une phase de caractérisation de comportements grâce à un algorithme de clustering suivi d'une phase d'analyse de comportements. L'étape de caractérisation est notamment menée par l'étude de deux descripteurs de l'évolution de clusters par rapport à leur barycentre. Contrairement à beaucoup

de travaux existants exploitant le clustering de données, cette technique s'émancipe du seuil qui détermine à partir de quelle distance un cluster est trop éloigné de la majorité des clusters représentant un comportement normal du système et devient un cluster qui traduit la présence d'une anomalie dans les données traitées. Nous appelons cette technique reposant sur l'étude de descripteurs *DESC*.

Dans nos travaux, nous mettons en œuvre ces entités sous forme de plusieurs modules hébergés sur une plateforme d'expérimentation. Nous déployons de plus un module d'injection de fautes ainsi que des cas d'étude de service cloud sur cette plateforme.

Le déploiement d'un module d'injection de fautes possède une double fonction. La première fonction est de construire des jeux de données d'entrainement pour des algorithmes à apprentissage supervisé avec des entrées représentant à la fois des comportements normaux et anormaux. C'est seulement avec un jeu de données comportant des entrées avec et sans anomalies que les modèles peuvent apprendre à classifier les entrées de nouveaux jeux de données lors d'une phase de détection. Cette technique nous permet de réaliser la collecte de jeu de données pour apprentissage supervisé de manière méthodique et dans un temps d'observation restreint. Nous savons ainsi précisément dans quels cas d'anomalies nos évaluations sont généralisables. De plus, sans l'injection de fautes plusieurs mois voire des années d'observation du système seraient nécessaires avant que des anomalies ne s'activent. La deuxième fonction concerne l'évaluation de notre stratégie de détection. L'injection de fautes nous permet d'émuler la présence d'anomalies en phase de détection sur notre plateforme expérimentale et dans un temps d'observation restreint.

Le déploiement des cas d'étude permet d'évaluer les performances de détection de notre stratégie. Cette évaluation est réalisée pour les différents fonctionnements de nos entités. Elle nous permet également d'identifier quels sont les tests à mener dans un but de validation rigoureuse de techniques de détection d'anomalies. Nous évaluons de plus la durée de validité des modèles de détection. Une telle évaluation n'est pas toujours effectuée dans les travaux de l'état de l'art.

Deux cas d'étude sont présentés dans ce manuscrit de thèse. Le premier est le système de gestion de base de données (SGBD) MongoDB. En 2016, MongoDB est le SGBD NoSQL le plus populaire et le quatrième SGBD le plus utilisé, tout modèle de base de données confondus [1]. MongoDB est un cas d'étude pertinent car il est implémenté pour le dimensionnement dynamique, la flexibilité d'utilisation ainsi que la rapidité d'accès aux données. Il est donc parfaitement adapté pour être un service cloud.

Le deuxième cas est une fonction réseau virtualisée et plus particulièrement un IP Multimédia Subsystem (IMS) appelé Clearwater [2]. Ce cas d'étude est représentatif des télécommunications et spécialement implémenté pour les clouds. Le récent intérêt des fournisseurs de télécommunications pour les clouds a amené au concept de fonction réseau virtualisée (ou en anglais Virtual Network Function, noté VNF).

1. http ://db-engines.com/en/ranking (vu octobre 2016)
2. http ://www.projectclearwater.org/about-clearwater/ (vu octobre 2016)

Les fournisseurs de télécommunications souhaitent en effet mettre à profit les bas coûts ainsi que la flexibilité de déploiement de plateformes à base de serveurs COTS utilisés dans les clouds en travaillant à la migration des fonctions réseau telles que les routeurs ou les pare-feux sur ces serveurs. Ces fonctions réseau ont en effet toujours été spécifiquement implémentées pour du matériel spécialisé (comme des routeurs ou pare-feu du fabricant CISCO). Lorsqu'elles sont adaptées au cloud, ces fonctions sont appelées VNFs.

Plan de la thèse

Ce manuscrit de thèse est structuré en six chapitres. Le début du manuscrit est dédié à la présentation du contexte de nos travaux ainsi que de l'état de l'art associé. Ainsi, le chapitre 1 présente les contextes du cloud computing et de la sûreté de fonctionnement dans lesquels s'inscrivent nos travaux. Le chapitre 2 présente un état de l'art des techniques de détection d'anomalies par apprentissage automatique.

Le cœur du manuscrit présente en détail une stratégie de détection d'anomalies que nous avons définie de manière à répondre aux contraintes de déploiement dans un cloud. Le chapitre 3 définie cette stratégie. Elle est fondée sur l'observation des comportements d'un système grâce à ses compteurs de performance. Le chapitre 4 décrit la mise en œuvre des modules nécessaires au déploiement des entités de notre stratégie sur une plateforme expérimentale.

Les deux derniers chapitres présentent les tests d'évaluation de performance menés pour l'application de notre stratégie de détection à deux cas d'étude représentatifs de services cloud. Le chapitre 5 présente les tests menés sur le SGBD MongoDB. Le chapitre 6 présente les tests plus approfondis menés sur la VNF Clearwater.

Enfin, la conclusion générale fait le bilan de nos travaux et présente des perspectives de recherche.

Chapitre 1

Contexte et concepts fondamentaux

Sommaire

1.1 Le cloud computing **9**
1.1.1 Définition et caractéristiques 10
1.1.2 La virtualisation 11
1.2 Problématique de sûreté de fonctionnement dans les clouds **12**
1.2.1 La sûreté de fonctionnement 12
1.2.2 Détection d'anomalie dans les services clouds 15
1.3 Conclusion **18**

Ce chapitre présente le contexte scientifique et technologique de nos travaux ainsi que leur problématique principale. Il décrit d'abord le contexte technologique traité qu'est le cloud computing. Il présente ensuite les concepts fondamentaux de la sûreté de fonctionnement, et les met en parallèle avec les besoins primordiaux dans le contexte spécifique des services déployés dans des clouds.

1.1 Le cloud computing

L'informatique dans les nuages, cloud computing ou *cloud*, est un paradigme marketing pour les opérateurs de centres de données. Il est né de la volonté de s'emanciper de l'achat et de la maintenance de matériel informatique spécialisé (comme proposé par les grilles informatiques [Foster 2008]), tout en n'ayant pas à planifier précisément des besoins en ressources informatiques mais en étant facturé à l'usage [Bauer 2012]. Les caractéristiques du cloud computing sont décrites plus en détail dans la suite de cette section. Nous présentons également les concepts fondamentaux associés qui seront utilisés dans nos travaux.

1.1.1 Définition et caractéristiques

Selon le National Institute of Standards and Technology (NIST) le cloud computing est *un modèle de délivrance par un accès réseau ubiquitaire, pratique et à la demande, d'un ensemble de ressources informatiques partagées et configurables* [Grance 2012]. Les ressources informatiques mises à disposition par un cloud sont appelées *services cloud*. Afin d'obtenir un service, un utilisateur s'adresse à un *fournisseur de service*. On dit que l'utilisateur peut alors *consommer* le service cloud demandé. Il peut être noté qu'un fournisseur peut lui-même potentiellement dépendre de services cloud tiers pour fournir ses propres services.

L'appellation de cloud provient de la représentation historique des réseaux, et particulièrement de l'Internet, par un nuage. La représentation en forme de nuage des réseaux permet de masquer une architecture complexe d'interconnexion de réseaux lorsque celle-ci n'est pas l'intérêt de l'étude en cours. De la même manière, un cloud se représente par un nuage afin de masquer sa complexité. Le modèle de cloud computing provient en effet lui-même de plusieurs paradigmes informatiques complexes tels que les grappes de serveurs et les grilles informatiques.

Le cloud est notamment défini par cinq caractéristiques essentielles énumérées par le NIST [Grance 2012].

— **Libre service à la demande** : la demande d'accès à un service par un utilisateur est automatique et ne demande pas d'interaction avec le fournisseur de service.
— **Large accès réseau** : un réseau permet l'accès aux services. L'accès se fait de manière standard par des clients hétérogènes, légers ou lourds (téléphones mobiles, ordinateur personnel et stations de travail).
— **Mise en commun de ressources** : un fournisseur met en commun les ressources qu'il souhaite mettre à disposition de ses utilisateurs. Les ressources sont accessibles sur la base d'une architecture multi-entités. Elles peuvent être physiques ou virtuelles et sont allouées dynamiquement à des utilisateurs sous formes de services de diverses configurations (par exemple il est possible de sélectionner la configuration mémoire, processeur (ou unité centrale de traitement, UCT, en anglais central processing unit, CPU), ou réseau avec la bande passante). L'utilisateur n'a généralement pas la visibilité sur la localisation des ressources qu'il utilise. Il peut toutefois avoir un contrôle sur le site géographique du centre de données hébergeant un service demandé.
— **Élasticité** : les ressources peuvent être fournies et réattribuées de manière rapide et automatique selon la demande des utilisateurs.
— **Facturation à l'usage** : l'utilisation des ressources est surveillée, contrôlée et reportée aux utilisateurs et aux fournisseurs.

L'ensemble des ressources matérielles et logicielles permettant de mettre en œuvre ces cinq caractéristiques est appelée *infrastructure cloud*.

Les services cloud peuvent être délivrés selon trois modèles définis par le NIST. Ces modèles dépendent du niveau d'abstraction des ressources délivrées par le ser-

vice. Le plus bas niveau d'abstraction est la couche physique et le plus haut niveau est la couche applicative. Les modèles sont présentés ci-dessous. Dans chaque cas, les utilisateurs ont accès aux services cités sans autre accès au reste de l'infrastructure cloud.

— **Software as a Service (SaaS).** Le fournisseur met à disposition des utilisateurs des applications. Les utilisateurs ont accès aux points de configuration de l'application.
— **Platform as a Service (PaaS).** Le fournisseur met à disposition des outils et langages de programmation afin que les utilisateurs puissent déployer des applications.
— **Infrastructure as a Service (IaaS).** Le fournisseur met à disposition des ressources informatiques élémentaires telles que la CPU, la mémoire, le stockage et le réseau. Les utilisateurs peuvent alors déployer des systèmes d'exploitation et applications. Ils peuvent dans certains cas avoir aussi l'accès à des briques élémentaires de réseau telles que des parefeux.

1.1.2 La virtualisation

Nous présentons dans la suite un concept nécessaire à la compréhension de nos travaux : la virtualisation. En effet, le cloud repose sur la mutualisation de ressources pour plusieurs utilisateurs rendue possible grâce à la virtualisation. Sa description nous permet donc de mieux comprendre le cloud et les approches proposées dans nos travaux pour assurer la sûreté de fonctionnement.

La virtualisation a vu son intérêt mûrir dans les années 70 [Goldberg 1974] à la suite de l'apparition des architectures multiprocesseur et de la stratégie d'ordonnancement multitâche [Buzen 1973]. Ces nouvelles architectures ont nécessité de nombreuses études sur la garantie de l'intégrité des systèmes multiprocesseur et/ou multitâche dont les processus partageant mémoire et autres ressources pouvaient alors se corrompre entre eux. Les systèmes virtualisés proposent un moyen logiciel pour faire tourner sur des ressources matérielles partagées plusieurs systèmes dont les instructions sont interprétées et filtrées par un gestionnaire de machines virtuelles (en anglais Virtual Machine Monitor, noté VMM). La virtualisation intéresse alors aussi de par sa flexibilité pour tester et reconfigurer des applications ou des systèmes d'exploitation [Goldberg 1973].

De manière plus détaillée la virtualisation du matériel permet de segmenter des ressources pour leur utilisation partagée par plusieurs instances de systèmes indépendants appelés machines virtuelles (ou en anglais Virtual Machine, noté VM). Le VMM, aussi appelé *hyperviseur*, est chargé d'arbitrer les accès des VMs au matériel. Avant les années 2000, la virtualisation était réservée à un déploiement sur des serveurs aux ressources matérielles spécialisées. Fin 2005 et début 2006, Intel et AMD ont respectivement introduit des technologies de virtualisation pour processeur x86. La virtualisation est alors mise à disposition de serveurs *pris sur étagère* (en anglais commercial off-the-shelf ou COTS) et différents utilisateurs pouvaient désormais partager les ressources d'un même serveur tout en dépendant potentiellement de

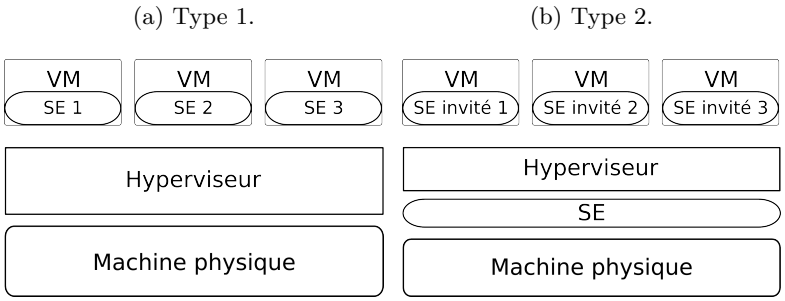


FIGURE 1.1.1 – Principaux types d'hyperviseurs.

systèmes d'exploitation différents et s'exécutant de manière isolée.

Deux types d'hyperviseurs prédominants se différencient selon s'ils segmentent directement ou indirectement les ressources physiques d'un serveur. Les hyperviseurs de type 1 (ou en anglais bare metal) s'appuient uniquement sur le micrologiciel (BIOS ou UEFI) du serveur hôte. Une représentation est donnée figure 1.1.1 (a). Les hyperviseurs de type 2 (ou en anglais hosted) s'exécutent à partir de l'environnement d'un système d'exploitation (SE) classique hôte. Une représentation est donnée figure 1.1.1 (b).

Dans le cas de serveurs de clouds, des hyperviseurs de type 1 sont employés afin de minimiser le nombre d'appels systèmes intermédiaires filtrés par un potentiel SE hôte intermédiaire.

1.2 Problématique de sûreté de fonctionnement dans les clouds

Le monde de l'informatique de nos jours est caractérisé par la rapidité de développement, le parallélisme et de la diversité des tâches à effectuer par les applications et services. Comme tout système informatique, le cloud n'est pas exempt des problématiques de sûreté de fonctionnement qu'il est nécessaire de traiter. Nous présentons dans cette partie les termes et concepts du domaine de la sûreté de fonctionnement tels que définis dans par [Laprie 1995]. Nous citons plus particulièrement les attributs, les entraves et les moyens de la sûreté de fonctionnement. Nous décrivons également la problématique de sûreté de fonctionnement et plus particulièrement de détection d'erreurs ou d'autres anomalies dans un cloud.

1.2.1 La sûreté de fonctionnement

La sûreté de fonctionnement d'un système est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre (qui peut être également un service cloud). Le service délivré par un système est son

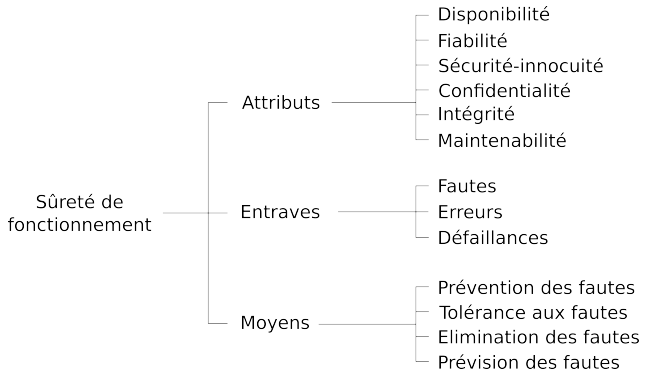


FIGURE 1.2.1 – Arbre de la sûreté de fonctionnement [Avizienis 2004]

comportement tel que perçu par son, ou ses utilisateurs ; un utilisateur est un autre système (humain ou physique) qui interagit avec le système considéré. L'exécution d'un système est perçue par son, ou ses utilisateurs comme une alternance entre deux états du service par rapport à l'accomplissement de la fonction du système : *service correct* (la fonction du système est accomplie) et *service incorrect* (la fonction du système n'est pas accomplie).

D'autres définitions de base vont avec les termes qui viennent d'être définis. Il est important de définir que le service délivré est le comportement du système tel que perçu par un utilisateur. Le *comportement* du système est ce que le système fait. Ce qui lui permet de faire ce qu'il fait est sa *structure*. Le comportement d'un système et sa structure peuvent avoir des états. Un *état* est une condition d'être par rapport à un ensemble de circonstances. Les autres systèmes ayant interagi ou interféré, interagissant ou interférant, ou susceptibles d'interagir ou d'interférer avec le système considéré composent l'*environnement* du système.

Finalement, la sûreté de fonctionnement est régie selon trois notions : ses *attributs* définissent un fonctionnement sûr, les *entraves* décrivent les circonstances indésirables (correspondent aux causes ou résultats de la non sûreté de fonctionnement du système), et les *moyens* contribuent à assurer un fonctionnement sûr en dépit des entraves. Ces classes sont détaillées dans la suite et sont représentées figure 1.2.1.

1.2.1.1 Attributs de la sûreté de fonctionnement

La sûreté de fonctionnement peut être vue selon des propriétés différentes mais complémentaires qui définissent ses attributs :

— le fait d'être prêt à l'utilisation conduit à la *disponibilité*,
— la continuité du service délivré conduit à la *fiabilité*,
— le fait d'être sans danger pour l'environnement conduit à la *sécurité-innocuité*,
— la non-occurrence de divulgations non-autorisées de l'information à des personnes non-autorisées conduit à la *confidentialité*,

— la non-occurrence d'altérations inappropriées de l'information conduit à l'*intégrité*,
— l'aptitude aux réparations et aux évolutions conduit à la *maintenabilité*.

La disponibilité, la confidentialité, et l'intégrité vis-à-vis des actions autorisées constituent la *sécurité-immunité*.

1.2.1.2 Entraves à la sûreté de fonctionnement

On distingue trois types d'entraves : les défaillances, les erreurs et les fautes. Une défaillance survient lorsqu'un service délivré dévie de l'accomplissement de sa *fonction*. Lorsqu'un service délivré dévie de l'accomplissement de sa fonction, il y a *défaillance du système* (ou *défaillance*). Une *erreur* est la partie de l'état d'un système qui est susceptible d'entrainer une défaillance. Par propagation, plusieurs erreurs peuvent être créées avant qu'une défaillance ne survienne. La cause adjugée ou supposée d'une erreur est une *faute*.

Les sources des fautes sont extrêmement diverses. Elles peuvent être classées selon cinq points de vue : leur cause phénoménologique, leur nature, leur phase de création ou d'occurrence, leur situation par rapport aux frontières du système et leur persistance. Nous ne décrivons pas ici les fautes qu'il est possible de distinguer dans chacun de ces points de vue. La considération simultanée des points de vue donne des fautes combinées. Un exemple de faute combinée est le suivant : une faute peut être due à l'homme, accidentelle (non intentionnelle), introduite lors du développement du système, interne (qui ne résulte pas des interactions du système avec son environnement) et temporaire (liée à des conditions internes ou externes ponctuelles, et donc présente pour une durée limitée). Certaines combinaisons ne sont toutefois pas vraisemblables (par exemple, une faute ne peut pas être physique et intentionnelle).

Cinq étiquettes permettent de regrouper l'ensemble des fautes combinées vraisemblables : les fautes physiques, les fautes de conception, les fautes d'interaction, les fautes logiques malignes et les intrusions.

1.2.1.3 Moyens pour la sûreté de fonctionnement

Afin qu'un système soit sûr de fonctionnement en dépit des potentielles entraves, des moyens pour la sûreté de fonctionnement doivent être mis en œuvre et utilisés de manière combinée. Ils sont au nombre de quatre. Tout d'abord la *prévention des fautes* qui vise à empêcher l'occurrence ou l'introduction de fautes dans un système. Elle relève de l'ingénierie générale des systèmes. Ensuite la *tolérance aux fautes* qui permet à un système de délivrer sa fonction en dépit des fautes existantes. La tolérance aux fautes est mise en œuvre par le *traitement d'erreur* et par le *traitement de fautes*. Le traitement d'erreur est destiné à éliminer les erreurs, si possible avant qu'une défaillance ne survienne. Le traitement de faute est destiné à éviter qu'une ou plusieurs fautes ne soient activées à nouveau. L'*élimination des fautes* œuvre à réduire la présence des fautes aussi bien en nombre qu'en sévérité de leurs impacts

potentiels sur le fonctionnement du système. Enfin, la *prévision des fautes* vise à estimer la présence, la création et les conséquences des fautes.

Nos travaux considèrent en particulier la tolérance aux fautes par traitement d'erreur. Ce traitement fait appel à trois primitives qui sont la *détection d'erreur* qui vise à déceler un état erroné d'un système, le *diagnostic* qui estime les dommages créés par les erreurs détectées, et le *recouvrement d'erreur* qui substitue un état exempt d'erreur à l'état erroné.

Nous focalisons principalement nos travaux sur la détection d'erreur. Dans les systèmes ou applications complexes, la constatation de défaillance par l'administrateur du système n'est pas directe. Elle peut nécessiter plusieurs analyses concernant la caractérisation du comportement du système. Ainsi nous travaillons en particulier à détecter des erreurs mais aussi les manifestations de défaillances dans des données liées au système étudié (par exemple dans des compteurs de performance système caractérisant l'utilisation des différentes ressources du système). Nous appelons cette détection *détection d'anomalie*. Une anomalie se traduit par une variation significative de ces données par rapport à une situation de fonctionnement normal.

1.2.2 Détection d'anomalie dans les services clouds

Parmi les nombreuses problématiques issues du domaine de la sûreté de fonctionnement des clouds, nous traitons dans ce manuscrit une problématique liée aux attributs de disponibilité et de fiabilité. Ces propriétés sont en effet primordiales pour les fournisseurs de services cloud, notamment de par leur engagement de service à la demande. Assurer ces deux propriétés pour des services à la fois différents, en allocation dynamique et pour des utilisateurs aux demandes hétérogènes (ils peuvent avoir différentes attentes de disponibilité ou bien différents besoins en termes de ressources) représente un défi pour les fournisseurs. Ce défi est d'autant plus important que les utilisateurs demandent à ce que les services rendus soient au moins aussi sûrs de fonctionnement que le déploiement local d'applications traditionnelles le serait.

Dans cette section nous présentons les caractéristiques de ce défi ainsi que les critères de sélection d'une technique de détection d'anomalies adaptée à ces caractéristiques. Nous nous plaçons dans le contexte où la détection est réalisée *dans les services cloud* en prenant le *point de vue fournisseur de services cloud*.

1.2.2.1 Caractéristiques de la détection d'anomalies dans les services cloud

Nous identifions trois caractéristiques au défi de la détection d'anomalies dans des services cloud :

— le type d'anomalies à détecter,
— le type de données traitées,
— les moyens humains et financiers mis en œuvre.

Type d'anomalie à détecter. À l'heure des services cloud toujours plus variés, il est difficile d'assurer qu'un service est exempt de faute. La concurrence mondiale dans le domaine amène à des services mis sur le marché alors que les phases de tests n'ont pas nécessairement pu tester toutes les fonctions à délivrer ou bien tester la compatibilité des services avec le matériel (dans le cas d'un matériel virtualisé, la compatibilité à assurer est la même que dans le cas où il ne l'est pas). Des défaillances peuvent donc potentiellement survenir. Ces défaillances peuvent avoir pour cause des fautes aussi diverses que les fautes rencontrées dans les systèmes traditionnels mais également des fautes liées à la virtualisation des services et à l'orchestration de ces services.

De manière générale, les différents types d'anomalies restent les mêmes que ceux représentés par les cinq classes de fautes présentées dans les entraves à la sûreté de fonctionnement 1.2.1.2.

Dans notre contexte, les fournisseurs de services cloud ne ciblent pas la détection d'anomalies liées à des fautes de conception des applications déployées par les utilisateurs. Ils ciblent les fautes liées aux services et systèmes qui sont directement sous leur contrôle.

Données traitées. La détection d'anomalies dans un système informatique peut être menée par exemple par l'analyse de la mémoire du système, l'introspection de données reçues ou émises (sur un réseau ou sur un bus par exemple), l'étude des processus actifs ou l'analyse de données de surveillance système obtenues par le biais de sondes ou de logs. En particulier, le traitement de ces données peut permettre de détecter la présence d'anomalies dans des clouds.

Du point de vue fournisseur, ces données sont disponibles en masse pour tous les services de leur centre de données. L'ère des masses de données n'est pas nouvelle, mais de nos jours, elle relève surtout de la numérisation de chacune des données disponibles d'un système et de ses sous-systèmes, ainsi que des nouvelles techniques de traitement de ces données [Schroeck 2012]. Nous citons ici quelques exemples de domaines divers collectant quotidiennement des masses de données de type différents dont les applications peuvent être hébergées par des services cloud : les entreprises de télécommunication collectent des millions de logs d'appels par jour et les conservent plusieurs mois ; les grandes plateformes de vente en ligne traitent des milliers de connexions par jour de clients différents ; les sites de réseaux sociaux publient des millions de messages par jour ; les météorologistes fondent leurs travaux sur l'information de milliers de capteurs météorologiques de tout type qu'ils analysent par la suite ; les centres financiers réalisent des millions d'opérations par minute basées sur l'analyse de milliers d'indicateurs financiers, etc.

Moyens mis en œuvre. Les moyens humains et financiers mis en œuvre pour la détection d'anomalies résultent de choix entre le coût de déploiement et l'efficacité de la technique de détection. Néanmoins, le contexte du cloud avec son automatisation de la délivrance de services amène les fournisseurs à surtout limiter le nombre

et la complexité des actions humaines nécessaires à la détection en automatisant le contrôle. En effet, le but est d'obtenir un contrôle autonome et fonctionnel en continu.

1.2.2.2 Critères pour la sélection d'une technique de détection d'anomalies

Nous présentons maintenant les critères de sélection d'une technique de détection d'anomalies afin qu'elle soit efficace et pertinente vis-à-vis des caractéristiques identifiées.

Quatre critères sont définis, dont les trois premiers découlent de la caractéristique visant à minimiser le nombre et la complexité des actions humaines durant la détection.

Premièrement, il est nécessaire que la détection d'anomalies prenne en compte l'évolution d'un système dans le temps. Cette évolution peut se constater suite à des reconfigurations, par exemple de ressources d'applications ou de serveurs, ou bien à la suite de changements du nombre et du type de requêtes reçues par le système (appelés *charge de travail*). Des reconfigurations ou bien des changements de charge de travail font que les services cloud possèdent plusieurs comportements légitimes d'exécution qu'il serait fastidieux de recenser manuellement.

Deuxièmement, afin de pouvoir traiter le premier point, le caractère complexe des services cloud est important à souligner. Dans des systèmes de contrôle/commande dont la ou les applications sont connues à l'avance, il est possible de connaître les caractéristiques de l'application a priori. Il est donc aussi possible de déduire des ensembles d'états de fonctionnement et de reconnaitre des situations anormales. Plus un système est grand avec des tâches variées voire inconnues a priori, plus il est difficile de dissocier un état de comportement normal d'un état de comportement anormal. Pour cette raison, il est nécessaire de faire appel à des techniques automatiques définissant des comportements (normaux ou anormaux) d'un système. Cette automatisation de l'analyse des masses de données, et donc de la détection d'anomalies, passe par l'utilisation d'outils provenant de plusieurs domaines de la recherche scientifique. Ces domaines sont l'informatique avec les techniques d'*apprentissage automatique* (aussi appelé de manière plus courte *apprentissage* ou en anglais *machine learning*), les statistiques, ou les théories des probabilités et de l'information.

Troisièmement, afin d'optimiser encore l'effort de déploiement et de paramétrage d'une tâche de détection d'anomalies pour de nombreux services cloud, il parait judicieux de sélectionner des techniques s'appliquant à tous les services d'un fournisseur, peu importe leur fonction et leur implémentation.

Quatrièmement, la détection d'anomalies appliquée dans notre contexte doit traiter les données en phase opérationnelle (dit traitement *en ligne*). La détection en ligne doit se réaliser soit en amont de l'occurrence d'une anomalie grâce à la détection de ses symptômes préliminaires, soit directement après que l'anomalie a affecté le système étudié. Ceci permet de réagir afin de traiter et d'éliminer l'anomalie avant qu'elle ne provoque l'interruption d'un service. De cette nécessité découle

également le besoin d'optimisation des temps de traitement des données générées par les clouds.

1.3 Conclusion

Dans ce chapitre nous avons présenté le paradigme du cloud computing. Un cloud est constitué de centres de données délivrant via l'Internet des services informatiques disponibles à la demande et facturés à l'usage.

Comme dans tout système informatique, la sûreté de fonctionnement est la problématique majeure des clouds. Nous avons donc introduit les concepts fondamentaux de cette discipline. Nous avons également présenté une problématique particulière qu'est la détection d'anomalies dans le cas des services cloud d'un point de vue fournisseur. Pour cela nous avons mis les concepts fondamentaux en parallèle des besoins des services cloud pour la détection d'anomalies.

La détection d'anomalies dans les services cloud en particulier est un défi pour les fournisseurs de service. Afin de traiter ce défi, nous avons défini quatre critères permettant la sélection d'une technique de détection d'anomalies adaptée à ce défi, dont les trois premiers découlent du besoin de minimiser les actions humaines lors de la détection.

1. La détection d'anomalies doit prendre en compte les changements de charge de travail et les reconfigurations possibles des services cloud.
2. La complexité des services fait que la première caractéristique énoncée ne peut être assurée qu'avec des techniques de traitement automatique des données.
3. La technique de détection doit être applicable à tout type de service cloud.
4. La détection doit de plus se faire en ligne, en continu et soit en amont soit très rapidement après l'occurrence d'une anomalie.

Le chapitre suivant établi un état de l'art des travaux traitant de la détection d'anomalies par apprentissage.

Chapitre 2

Détection d'anomalie par apprentissage - État de l'art

Sommaire

2.1 Détection d'anomalie **20**
2.1.1 Différents types de données 20
2.1.2 Techniques basées sur les statistiques, les théories des probabilités et de l'information 22
2.1.3 Techniques de détection basées sur l'apprentissage automatique 23
2.2 Évaluation expérimentale de performances de détection **29**
2.2.1 Injection de fautes 30
2.2.2 Métriques d'évaluation 31
2.3 Conclusion **34**

Ce chapitre présente une étude synthétique des techniques de détection d'anomalies dans de grands systèmes informatiques et dans les clouds en particulier. Les principaux concepts associés sont de plus définis. La détection d'anomalies dans les services cloud est en effet un domaine récent qui est surtout fondée sur les techniques de détection dans les grands systèmes informatiques en général. Ces travaux sont donc complémentaires afin d'avoir un aperçu des techniques envisageables pour la détection d'anomalies dans les services cloud respectant les critères que nous avons définis.

La détection d'anomalies est souvent menée à bien grâce à des techniques d'apprentissage automatique et de statistiques. Plusieurs travaux illustrant ces techniques sont donc présentés. Les différents cas d'application et techniques de la détection d'anomalies sont de plus répertoriés dans plusieurs études [Chandola 2009, Salfner 2010]. Dans ce chapitre néanmoins, une attention particulière est donnée à la description des techniques utilisant l'apprentissage automatique qui est le domaine exploité dans nos travaux.

Ce chapitre est structuré en deux sections présentant respectivement des techniques de détection d'anomalies et des méthodes et mesures d'évaluation de ces

techniques dans un contexte expérimental.

2.1 Détection d'anomalie

La littérature liée au cloud regroupe un certain nombre de travaux qui se penchent particulièrement sur les problèmes de gestion de ressources élastiques des VMs ou des services [Kundu 2012, Matsunaga 2010, Bodík 2009, Silvestre 2015a], de reconfiguration de serveurs [Cerf 2016], de gestion de ressources et d'énergie des serveurs [Chase 2001, Berral 2010] et de sécurité [Bhat 2013, Gander 2013]. Des travaux existent également sur des outils ou *frameworks* dédiés au traitement en général de données en masse provenant d'infrastructures cloud, comme l'étude dans [Pop 2016]. Peu d'études ont été dédiées à la détection d'anomalies en général et sans se focaliser sur un type d'anomalie en particulier (comme les attaques par le réseau). Ainsi nous présentons ici autant de travaux liés en particulier au cloud que de travaux sur de grands système informatique peu importe leur paradigme de déploiement.

Les travaux traitant de la détection d'anomalies regroupent sous une même appellation des techniques de détection de comportements anormaux et d'états erronés d'un système. Selon les domaines de recherche et les cas d'étude, les *anomalies* sont dénommées de différentes manières telles que données aberrantes, exceptions, attaques, erreurs ou violation de service.

La détection d'anomalies se définit usuellement par l'action de discriminer dans un jeu de données composé d'*observations* (i.e. les lignes) et d'*attributs* (i.e. les colonnes) non connu à l'avance caractérisant un système cible, des observations qui ne correspondent pas à la tendance globale représentée par la majorité des observations [Chandola 2009]. La difficulté de la tâche est d'identifier précisément cette tendance globale.

Ces méthodes se différencient notamment entre elles de par les données traitées pour détecter des anomalies ainsi que les algorithmes de détection utilisés. Les algorithmes de détection proviennent des domaines cités dans le chapitre précédent à savoir les statistiques, les théories des probabilités et de l'information, et l'informatique. Nous présentons donc des méthodes de détection d'anomalies classifiées selon la nature des données utilisées pour la détection dans un premier temps, puis selon les algorithmes mis en œuvre dans un second temps. Nous présentons dans un troisième temps certains concepts de l'apprentissage automatique afin d'introduire des méthodes de détection basées sur ce domaine. En effet, l'apprentissage automatique est le domaine directement lié à nos contributions principales.

2.1.1 Différents types de données

Les données à traiter pour la détection d'anomalies dans un système informatique peuvent être de trois grands types.

Premièrement, nous trouvons des journaux d'historique d'évènements (ou *logs*), systèmes ou applicatifs comme utilisés pour la détection d'anomalies des travaux

[Salfner 2007, Watanabe 2012, Xu 2009, Liang 2006]. Ces journaux regroupent des évènements survenus respectivement au niveau des ressources d'un système ou correspondant aux opérations des tâches d'une application. Les logs systèmes fournissent une vue globale d'une machine. Leur analyse dépend du système d'exploitation de la machine. Les logs applicatifs incluent des évènements générés par l'exécution d'une application et dépendent le plus souvent de l'implémentation de celle-ci lorsque qu'elle n'est pas standardisée.

Deuxièmement, la détection peut également se faire à partir des traces d'audit qui regroupent des évènements de haut niveau correspondant à des échanges sur un réseau ou à des actions exécutées par des utilisateurs, comme utilisés pour la détection d'anomalies des travaux [Denning 1987, Lee 1999, Heberlein 1995, Mukherjee 1994]. Selon le niveau d'observation (paquets réseau ou bien actions utilisateurs) le format des traces d'audit est plus ou moins dépendant de l'environnement de déploiement.

Les logs ainsi que les traces d'audits sont des données qu'il est nécessaire de décomposer analytiquement (en anglais *parser*) avant leur utilisation pour la détection. L'exploitation et le traitement des logs et traces d'audit pour des études de sûreté de fonctionnement n'est pas toujours facile comme illustré dans plusieurs travaux [Simache 2002, Simache 2001]

Troisièmement, certaines méthodes de détection traitent des statistiques d'utilisation d'une application ou bien d'observations de performances système (en anglais *system monitoring* ou de manière plus courte dans cette thèse *monitoring*). Les statistiques d'utilisation d'une application sont par exemple les indicateurs de performances système utilisées par cette application ou des compteurs (compteurs de ressources, d'erreurs, etc.). Ces données sont dépendantes de l'implémentation de l'application. Les observations de monitoring sont par exemple la CPU (Central Process Unit) utilisée, le nombre de paquets IP (Internet Protocol) envoyés par seconde, le statut d'un disque, ou le nombre de rotations d'un ventilateur comme utilisées dans [Dean 2012, Tan 2012, Zhang 2013, Guan 2012a, Nguyen 2013]. Ces observations fournissent une vue globale d'une machine. Des travaux tels que [Farshchi 2015a] utilisent à la fois des logs ainsi que des observations de monitoring pour détecter des anomalies. Dans ce cas, la détection porte sur des VMs s'étégnant pour cause de défaillances ou suite à une extinction légitime. La détection profite donc sur une représentation plus complète d'un système mais elle est amenée à être plus longue à exécuter.

Dans nos travaux nous nous sommes particulièrement intéressés à ce dernier type de données pour la détection. Ce choix a principalement été motivé par la variété d'applications pouvant être mises en œuvre dans des services cloud et le besoin d'être le moins dépendant possible des spécifications de ces services.

Dans un contexte de détection pour une application particulière, les données au plus proche de l'application sont plutôt convoitées. Elles amènent en effet davantage de connaissance sur les comportements rencontrés.

2.1.2 Techniques basées sur les statistiques, les théories des probabilités et de l'information

Dans ces travaux, le modèle de comportement normal d'un système est décrit par des statistiques ou un modèle de probabilité représentatif des données collectées sur le système et le décrivant.

Tiresias [Williams 2007] est un système de prédiction de défaillance en ligne pour des systèmes distribués. Il fonctionne selon une approche boîte noire (c'est-à-dire en ne connaissant que les spécifications fonctionnelles d'un système). Il suppose qu'une défaillance est précédée d'un comportement instable du système. Tiresias détecte des comportements instables dans les observations de monitoring du système en construisant des séries temporelles analysées grâce à des seuils de détection et par la technique de DFT (Dispersion Frame Technique). Des séries temporelles de performances système sont de plus analysées dans [Sharma 2013] par le biais de relations calculées deux à deux entre chaque métrique d'une observation. Une anomalie est détectée lorsque certaines de ces relations sont cassées (i.e. elles ne sont plus constatées à un instant donné) et localisée en analysant les métriques dont les relations sont cassées. Ces relations sont définies par un modèle autoregressif (dans lequel la valeur d'une série temporelle est expliquée par ses valeurs passées) avec des variables exogènes supposant que les métriques sont indépendantes entre elles.

Les travaux dans [Eskin 2000] opèrent une détection d'anomalies en utilisant deux distributions de probabilité sur des traces d'appels système UNIX afin d'évaluer par des tests statistiques si une suite de symboles contenus dans une trace correspond à une anomalie ou non. Des symptômes d'anomalies sont détectés de manière probabiliste dans les travaux de [Shen 2009]. Pour cela, des chutes de performance du système étudié sont prédites en comparant l'exécution d'un système à celle d'un autre système servant de modèle (un benchmark par exemple). L'exécution d'un système est définie sur la base de mesures telles que le débit de réponse à des requètes ou le nombre d'entrées/sorties concurrentes dans un système. Des mesures provenant de la théorie de l'information sont utilisées dans les travaux de Wenke et Dong [Lee 2001] afin de détecter des anomalies dans des captures de paquets réseau, des appels systèmes du serveur de messagerie *sendmail*, ainsi que dans des traces d'appels système UNIX. Les travaux [Wang 2010] présentent un framework nommé EbAT pour la détection d'anomalies en ligne. Il repose sur l'analyse des séries temporelles du calcul d'entropie de observations de monitoring. Il ne dépend pas de règles ou de modèles prédéfinis. Les auteurs précisent toutefois qu'EbAT ne peut être appliqué sur des métriques dont l'évolution dans le temps est très variable. [Salfner 2007] utilise une extension à temps continu des modèles de Markov cachés (i.e. le modèle recherché est un processus markovien dont les paramètres sont inconnus) dans le but d'anticiper des défaillances de machines. [Gong 2010, Nguyen 2013] détectent des anomalies à partir de observations de monitoring et en utilisant des transformées de Fourier rapides (en anglais Fast Fourier Transform ou FFT) ou des chaines de Markov selon si la charge de travail du système étudié peut être simplifiée par des modèles (en anglais patterns) d'exécution.

2.1.3 Techniques de détection basées sur l'apprentissage automatique

La détection d'anomalies se doit donc de discriminer une observation correspondant à une anomalie (i.e. une observation qui a été faite alors que le système subissait une anomalie) ou d'une observation de comportement normal. Lorsque la détection d'anomalies est mise en œuvre par des techniques d'apprentissage automatique, des algorithmes sont utilisés afin de réaliser cette discrimination, sur la base de précédentes observations. Elle est appelée *prédiction* [Alpaydin 2004]. Un algorithme d'apprentissage automatique fait une prédiction quant à la catégorie d'une nouvelle observation (i.e. qu'il n'a jamais eu à traiter avant), sur la base de précédentes observations déjà traitées. La catégorie d'une observation, par exemple la catégorie de comportement normal et la catégorie anomalie, est assignée à une observation grâce à une étiquette, aussi appelée *label* (souvent un nombre entier). La prédiction faite par l'algorithme peut être confirmée ou infirmée a posteriori par un opérateur.

Afin de lever une ambigüité éventuelle, nous précisons qu'il est commun de dire qu'un "algorithme prédit des labels" mais également qu'un "algorithme prédit des défaillances", dans le cas où celui-ci "détecte des *symptômes préliminaires*" à une défaillance.

Dans le cas où un *modèle* peut être construit à partir d'observations labelisées dans le but de discriminer les observations de labels différents, on parle de *classification supervisée* de données [Michie 1994]. Les différents labels indiquent alors des *classes* d'appartenance. Les observations labelisés permettant de construire le modèle sont appelées *données d'entrainement*. Le modèle créé fait des prédictions quant à la classe de tout nouveau cas non connu (appelé *données de test*) et ne faisant donc pas partie du jeu de données d'entrainement. Une classification supervisée s'opère donc en deux phases que nous appelons *phase d'entrainement* et *phase de détection* dans notre cas où nous utilisons la classification pour la détection d'anomalies. Les prédictions sont faites sous forme de probabilités. Un *seuil de prédiction* est alors à définir afin d'évaluer à quel niveau une alarme est levée ou non (par exemple 0.5). Lorsqu'un modèle est construit pour discriminer des observations d'anomalie par rapport à des observations de comportement normal), on parle d'une classification *binaire*. Un classifieur peut également permettre de discriminer par exemple des anomalies de plusieurs types par rapport à un comportement normal ou plusieurs types de comportements normaux par rapport à une unique anomalie. On parle alors de classification *multi-classe*. Par simplicité, lorsque nous parlons d'une détection d'anomalies implémentée par un algorithme de classification, nous parlons de *détection binaire* ou de *détection multi-classe*.

Dans le cas où aucune donnée d'entrainement labelisée n'est utilisée pour la construction d'un modèle, on parle de *classification non supervisée* de données [Michie 1994]. Une telle classification peut s'opérer avec une phase d'entrainement sur des observations qui ne sont pas labelisées (pour calculer des distances ou des distributions par exemple). La seconde phase consiste en une phase de description

de données. La description est générale et différente selon l'algorithme utilisé. Elle permet souvent une fouille des données.

Le cas intermédiaire est appelé *classification semi supervisée* [Chapelle 2009, Fu 2011]. Le modèle est créé et fonctionne par la suite avec des cas connus et non connus. Le modèle s'exécute avec les mêmes phases qu'un classifieur supervisé. Différents algorithmes pour cette classification existent. Certains fonctionnent par exemple à la manière d'une classification non supervisée avec un ajout de certaines contraintes, ou, correspondent à une classification supervisée à laquelle on ajoute de nouveaux exemples connus après la phase d'apprentissage. Dans les deux cas, le modèle créé cherche à s'adapter à la structure des données à partir d'un jeu de données d'apprentissage mais aussi en tenant compte de données de test qui lui arrivent au fur et à mesure de sa phase de détection. Les données non connues venant du jeu d'apprentissage sont utilisées pour le paramétrage du modèle tout en laissant certains degrés de liberté qui peuvent être modifiés par la suite de la phase de détection. Afin d'utiliser ces données, l'algorithme doit nécessairement reposer sur une ou plusieurs hypothèses permettant d'évaluer la classe probable d'appartenance d'un cas non connu.

Les deux types d'apprentissage que nous appliquons à la détection d'anomalies dans cette thèse à savoir un apprentissage supervisé et un apprentissage non supervisé sont présentés ci-dessous conjointement avec des travaux de détection les illustrant.

2.1.3.1 Détection par apprentissage supervisé

Plusieurs entraves à l'efficacité d'une procédure de classification supervisée peuvent exister. Nous faisons notamment mention ici du problème de complétude des données d'entrainement et du temps de collecte d'un tel jeu de données.

En effet, si nous voulons qu'un modèle soit apte à détecter des anomalies, celui-ci doit caractériser le système très précisément à la fois en comportement normal et en cas de présence d'anomalies. Les comportements normaux peuvent toutefois être multiples (comportement normal avec une charge de travail importante ou quasiment nulle par exemple), tout comme les comportements en présence d'anomalies (comportement en présence d'une machine qui a des accès disque trop longs ou comportement d'une machine qui a une bande passante quasiment nulle par exemple). Le grand nombre de comportements des systèmes informatiques (normaux et anormaux) amène la nécessité de travailler avec des méthodes pertinentes et rigoureuses de collecte et définition d'un jeu de données d'entrainement.

La collecte d'un jeu de données d'entrainement doit notamment être brève, qu'elle soit réalisée sur une plateforme expérimentale ou bien à partir d'un système réel. En effet, lors du déploiement initial du système, une collecte longue induit une longue période durant laquelle le système n'est pas surveillé car le système de détection d'anomalies n'est pas encore opérationnel. Une collecte longue suppose également que lors d'une reconfiguration de système, ou autre évolution du système, l'entrainement doit être reconduit sur une longue période, ce qui n'est pas

acceptable. Les jeux de données d'entrainement doivent donc être collectés dans une période restreinte de temps. Néanmoins, cela induit une vision nécessairement incomplète du système car représentative d'un temps d'exécution trop court de celui-ci. Cela est d'autant plus vrai que les anomalies sont des évènements rares comparés à des comportements normaux. Ainsi, plus un jeu de données d'entrainement est court, moins il contient d'anomalies, qui sont justement les éléments que l'on veut pouvoir discriminer efficacement. Nous rejoignons ici la nécessité de travailler avec des méthodes rigoureuses de définition d'un jeu de données d'entrainement. Plusieurs tests doivent ainsi être menés afin de confirmer que la taille du jeu de données d'entrainement permet des performances de détection acceptables.

Un autre défi réside dans la collecte d'un jeu de données de validation d'une technique de détection par apprentissage supervisé. La collecte peut être longue et faite sur une exécution de quelques mois voire un an par exemple du système. Dans ce cas il y a un risque d'une validation dont la valeur est obsolète due à l'évolution du système. La collecte d'un jeu de données de validation peut aussi être faite en même temps que la collecte du jeu de données d'entrainement. Dans ce cas précis, une seul jeu de données est collecté. Il est par la suite divisé en deux afin de constituer un jeu de données d'entrainement et un jeu de données de validation. Ce cas souffre quant à lui d'une vision nécessairement incomplète du système car représentative d'un temps d'exécution court du système.

Ces problématiques étant dites, l'attractivité des techniques par apprentissage supervisé réside notamment dans la rapidité de prédiction de la classe de nouvelles observations, aussi complexe soit le système (de l'ordre de la milliseconde en utilisant des arbres de décision par exemple). C'est en effet la phase d'entrainement des modèles qui est longue à aboutir.

Un autre avantage de ces techniques est qu'une prédiction peut de plus être faite de manière individuelle sur toute nouvelle observation récupérée périodiquement d'un système. Cette caractéristique peut être reprise pour l'utilisation en ligne de ces algorithmes. Un dernier aspect positif est que les performances de ces algorithmes dans les travaux que nous citons sont prometteurs. Tous les points cités font de la classification par apprentissage supervisé une technique largement utilisée dans les problèmes de détection d'anomalies.

Le choix de la procédure est laissé à l'utilisateur en fonction de ses contraintes de temps de prédiction ou bien de performance. Les les arbres de décision, les machines à vecteurs de support (en anglais Support Vector Machines, ou SVMs) et les réseaux de neurones supervisés sont des techniques courantes et testées dans des travaux dont le but est l'évaluation expérimentale de techniques populaires [Van Hulse 2007, Farshchi 2015b].

[Aleskerov 1997, Liang 2007, Duan 2009, Cohen 2004, Tan 2012, Zhang 2008, Tan 2010] sont des techniques de détection d'anomalies par apprentissage supervisé. Dans [Aleskerov 1997] les auteurs étudient des réseaux de neurones artificiels à apprentissage supervisé afin de détecter des fraudes sur cartes de crédit. Le travail de Liang et al. [Liang 2007] fait de la prédiction d'anomalie à partir des logs d'un IBM BlueGene/L en utilisant plusieurs classifieurs dont un qui repose sur la tech-

nique du plus proche voisin et en utilisant l'algorithme SVM. Ces classifieurs sont entrainés à détecter des symptômes précurseurs d'anomalies tels qu'un nombre trop important d'erreurs ou de warnings accumulés pour une tâche sur une fenêtre de temps. Fa [Duan 2009] est un système automatisant la détection d'anomalies grâce à une base de données de signatures de défaillances définies à partir de métriques de performance système. Fa repose sur CART (pour Classification and Regression Trees) et sur SVM. Le travail décrit dans [Cohen 2004] utilise des TAN (Tree-Augmented Bayesian Networks) créés à partir de données labelisées pour détecter des violations de service d'applications basées sur des web-services. Le travail issu de [Tan 2012] détecte des problèmes de performance système cloud en anticipant la valeur d'attributs par le biais de modèles de Markov dont les transitions du processus markovien dépendent de l'état courant ainsi que de l'état directement antérieur d'un attribut, et en classifiant les attributs par des réseaux bayesiens TANs. Dans [Zhang 2008], les forêts d'arbres décisionnels (en anglais Random Forest) sont utilisées afin de détecter des intrusions à partir du réseau. Enfin, ALERT [Tan 2010] est un système de prédiction d'anomalie fondé sur une étape de groupement appelée en anglais *clustering* permettant de faire des groupes de contextes d'exécutions (par exemple les groupes peuvent correspondre à différentes charges de travail) et sur une étape de création d'arbres de décisions entrainés à détecter des anomalies selon le contexte d'exécution identifié. Les modèles ont besoin d'être entrainés une unique fois et le système est déployable en ligne. De nouveaux modèles sont à entrainer lors de la détection de nouveaux contextes.

2.1.3.2 Détection par apprentissage non supervisé

Les techniques à apprentissage non supervisé visent à décrire des données. Elles permettent de découvrir des groupements d'observations similaires d'un jeu de données. Si l'on considère le clustering, le groupement est appelé *cluster*.

Les algorithmes les plus populaires sont les réseaux de neurones non supervisés, des modèles de régression, l'analyse en composantes principales ou bien le clustering. Ces algorithmes ont été utilisés dans plusieurs travaux pour la détection d'anomalies. Par exemple, la méthode UBL proposée dans [Dean 2012] pour la détection d'anomalies dans les clouds est fondée sur les réseaux de neurones non supervisés, avec l'hypothèse que des défaillances sont détectables grâce à des symptômes préliminaires observables dans des données de monitoring. Concernant les modèles de régression, [Cherkasova 2009] présente un outil permettant de détecter des changements de performance dans les applications et d'évaluer s'ils sont dûs à des anomalies ou bien à une augmentation ou réduction de la charge de travail en cours. Leur étude est fondée sur un modèle de régression construit à partir de l'étude de la consommation CPU d'un serveur ainsi que sur le nombre de transactions client (obtenu à partir de logs applicatifs). Une analyse en composantes principales est faite dans le travail [Xu 2009] qui porte sur une analyse de logs applicatifs et également dans [Lakhina 2004] pour la détection d'anomalies dans le trafic réseau.

Le clustering est l'algorithme dont nous donnons un court état de l'art dans la suite. Il a été par exemple utilisée dans eCAD [Zhang 2013], un outil de détection d'anomalies dans les données de monitoring de VMs d'un cloud ou dans [Leung 2005, Mazel 2011] pour la détection d'anomalies dans le trafic réseau. vNMF [Miyazawa 2015] [Niwa 2015] est un outil qui détecte des comportements anormaux dans les VNFs (Virtual Network Functions) fondé sur des réseaux de neurones non supervisés et une extraction de clusters à partir du modèle obtenu. vNMF exploite les données de monitoring de VNFs. Elle considère comme représentatifs d'anomalies, les clusters de petite taille et qui sont éloignés de tous les autres clusters. Dans ces deux derniers exemples, et comme généralement traité dans la littérature, les anomalies correspondent à des observations ne faisant partie d'aucun cluster.

Parmi les familles d'algorithmes de clustering nous citons les algorithmes hiérarchiques ou de partition. Les algorithmes hiérarchiques représentent les clusters de manière hiérarchique grâce à un dendrogramme où chaque étage représente un niveau de granularité différent identifiant les clusters. Selon si l'algorithme part d'un cluster unique qui est divisé de plus en plus à chaque niveau ou bien si l'algorithme utilise le principe inverse et commence par considérer chaque point comme représentant unique d'un cluster pour les regrouper selon un certain critère, on parle de méthode divisive ou agglomérative. Nous trouvons par exemple l'algorithme BIRCH [Zhang 1996] ou l'étude de Murtagh [Murtagh 1983] qui traitent de ces algorithmes.

Les algorithmes de partition identifient des partitions dans un jeu de données et non pas une structure de cluster en dendrogramme. Nous trouvons par exemple les travaux présentés dans [Ester 1996, Macqueen 1967, Beyer 1999]. Elles optimisent souvent un critère grâce à plusieurs itérations d'un algorithme.

Les algorithmes peuvent aussi être classés dans des familles différentes des deux que nous venons d'exposer. La diversité de familles envisageables est due à des techniques communes partagées par les algorithmes [Jain 1999]. Ces techniques sont par exemple basées sur l'identification des clusters par :

- le calcul de distance entre les observations d'un jeu de données (comme la technique du plus proche voisin [Beyer 1999]),
- le calcul de densité de points (comme Cassisi [Cassisi 2013], DBSCAN [Ester 1996], ou les algorithmes reposant sur DBSCAN [Esfandani 2010, Xiaoyun 2008]),
- l'utilisation d'une partie (comme CLIQUE [Agrawal 1998]) ou de tous les attributs d'un jeu de données dans les itérations de définitions des clusters (comme [Beyer 1999] et DBSCAN [Ester 1996]),
- l'association d'une observation à un cluster de manière tranchée ou selon un degré d'appartenance (on parle de groupement flou [Zadeh 1965]).

La similarité des points constituant un cluster peut être définie de nombreuses manières à savoir par des algorithmes reposant sur la distance entre les points, sur les zones de différentes densités de points ou bien sur des modèles génératifs [Aggarwal 2013b].

Considérant les algorithmes reposant sur les distances entre points, les clusters sont identifiés en minimisant la distance entre les points d'un même cluster

et en maximisant la distance entre les clusters, comme dans l'algorithme k-means [Macqueen 1967]. La densité peut aussi permettre de définir des clusters en définissant une densité cible pour les clusters et en évaluant de proche en proche les points dont l'entourage de points forme au minimum cette densité cible. Un exemple est l'algorithme DBSCAN [Ester 1996]. Un dernier type d'algorithme est de supposer un modèle génératif (par exemple un modèle de mélange gaussien) pour les données, d'associer par la suite une probabilité à chaque point pour ensuite conclure que les points de faible probabilité ne font partie d'aucun cluster caractérisé par une gaussienne et sont des anomalies.

Les algorithmes de clustering peuvent aussi être caractérisés par le type de données traitées (par exemple des données réseau, multimédia, texte ou catégorielles) et par le scénario d'acquisition des données. Certains algorithmes de clustering travaillent en effet sur des jeux de données de taille fixe, et d'autres sur des flux de données dans lesquelles les nouvelles observations arrivent périodiquement. Dans le premier cas, les clusters trouvés représentent entièrement le jeu de données. Dans le second cas, un groupe de clusters représente les données à un instant donné et une méthode de dévalorisation des clusters en fonction du temps doit être implémentée. En effet, un cluster peut représenter à l'instant t un flux de données et ne plus être représentatif à l'instant $t + n$. Des exemples d'algorithmes de clustering traitant des flux de données sont Clustream [Aggarwal 2003] ou l'algorithme proposé par Lühr [Lühr 2009], HPStream [Aggarwal 2004], Denstream [Cao 2006] ou D-stream [Tu 2009]. De nombreux algorithmes sont présentés dans l'étude de Aggarwal [Aggarwal 2013a].

Un même jeu de données peut donc être caractérisé par des clusters différents et l'expertise humaine vient souvent confirmer ou infirmer la pertinence d'un algorithme selon les données utilisées, selon le domaine d'étude et aussi selon le but de l'analyse.

Enfin, certaines techniques sont dites *hybrides* car elles utilisent plusieurs techniques et notamment par apprentissage supervisé et non supervisé. Un premier exemple est ALERT [Tan 2010] que nous avons présenté plus haut. Shon et Moon [Shon 2007] appliquent un algorithme SVM en mode supervisé et non supervisé afin de détecter des intrusions de systèmes à partir de l'analyse de trafic réseau. Guan et al. [Guan 2012b] utilisent une classification Bayésienne non supervisée pour détecter des anomalies et après la confirmation de ces anomalies, ils utilisent des arbres de décision à apprentissage supervisé pour détecter des défaillances dans des serveurs cloud.

2.1.3.3 Discussion sur les besoins de détection

De nos jours certains systèmes intègrent plusieurs mécanismes de tolérance aux fautes car ils ont été développés dans un contexte critique avec des exigences fortes de sûreté de fonctionnement. Ces systèmes incorporent donc des primitives de recouvrement d'anomalies qui peuvent elles-mêmes être divisées en trois types, à savoir la reprise (adoption d'un état antécédent stable du système), la poursuite (souvent

appelée "mode dégradé") et la compensation d'anomalie (évitement d'états erronés par utilisation de la redondance)[Laprie 1995]. D'autres systèmes, qu'ils aient été développés en ayant de moindre exigences de tolérance aux fautes ou bien qu'ils soient trop anciens pour intégrer des mécanismes nouveaux et plus efficaces, ne tolèrent que partiellement certaines fautes.

Ces deux catégories de systèmes ne sont pas amenées à gérer les anomalies de la même manière et avec les mêmes outils.

Dans le premier cas, en faisant l'hypothèse d'un système à haute redondance on peut imaginer que la non détection d'une anomalie peut ne pas avoir de conséquence catastrophique sur le système surtout grâce à la redondance qu'il intègre. Une telle hypothèse est plus forte, voire incorrecte, à faire sur un système appartenant à la deuxième catégorie de système présentée. Dans ce cas, le manquement à la détection d'une anomalie amène probablement à une coupure du service partielle ou totale pouvant être catastrophique.

Ainsi, dans le premier cas, une fausse alarme déclenchant un mécanisme de recouvrement d'anomalie par reprise ou poursuite pourrait être plus coûteuse financièrement de par un temps de coupure prolongé du service que l'activation seule de la compensation d'anomalie (supportée par la redondance du système). En effet la compensation d'anomalie peut s'exécuter automatiquement sans détection préalable.

Dans le second cas, il peut être plus pertinent de procéder à un recouvrement par reprise ou poursuite sur détection d'une anomalie, qu'elle s'avère être exacte ou non. En effet, il peut être moins coûteux de procéder à des recouvrements sans présence de faute (suite à une fausse alarme) que de prendre le risque de manquer la détection d'une anomalie menant à une défaillance catastrophique.

Lors de la validation d'une technique de détection, il est donc très important de bien choisir les mesures d'évaluation qui vont attester du bon fonctionnement de la technique.

2.2 Évaluation expérimentale de performances de détection

Dans cette section nous présentons une synthèse des besoins pour l'évaluation expérimentale des techniques de détection d'erreur.

L'évaluation des techniques de détection, amène à utiliser deux grandes classes de données : des données issues de l'exploitation opérationnelle comportant des erreurs reportées par les utilisateurs, et des données provenant de d'expérimentations contrôlées dans lesquelles des erreurs sont provoquées délibérément dans un système cible observé. Le second cas est celui qui nous intéresse dans nos travaux. Nous nous sommes donc intéressés à la manière de reproduire ces dernières sur une plateforme de test à l'environnement maîtrisé. Une telle tâche est appelée *injection de fautes*. Cette méthode permet d'injecter des fautes dans un système tout en observant celui-ci durant l'injection (les dates de commencement d'injection étant

préparées dans un protocole d'injection). Cette méthode est présentée dans cette section.

Enfin, une grande partie du travail concernant des approches de détection est d'analyser les résultats de détection et de comparer ces résultats à d'autres approches. Les techniques d'évaluation des performances de ces approches sont donc importantes et les plus connues d'entre elles sont présentées dans ce chapitre.

2.2.1 Injection de fautes

L'*injection de fautes* a pour objectif d'observer le comportement d'un système dans un état erroné ainsi que dans un environnement connu. Elle permet d'identifier et de comprendre les états potentiels d'un système menant à des défaillances. Pour cela des fautes sont insérées dans la structure du système pour émuler différents types d'erreurs susceptibles d'affecter le comportement du système et pour étudier leurs impacts. L'étude récente de [Natella 2016] présente en détail les concepts et domaines d'application de cette technique. Cette méthode peut notamment être utilisée comme procédure de test du système cible ou bien de mécanismes de tolérance aux fautes. Dans les deux cas, l'injection doit être non-intrusive de manière à ne pas biaiser les résultats de test. Les fautes peuvent être injectées lors de la simulation du système cible ou bien sur un prototype. Sur prototype, les fautes peuvent être injectées à deux niveaux : au niveau matériel grâce à une infrastructure spécialisée ou au niveau logiciel grâce à des altérations de code ou de données [Hsueh 1997, Arlat 1990]. Toutes ces méthodes sont complémentaires et possèdent des avantages mais aussi des inconvénients.

La simulation d'un système permet d'analyser l'impact des fautes injectées et leur propagation à différents niveaux de granularité. La simulation est également utilisée pour estimer des mesures de sûreté de fonctionnement telles que le temps moyen entre défaillances ou la disponibilité. La principale difficulté est liée à des temps de simulation très longs qui sont nécessaires pour obtenir des mesures précises. Des hypothèses sont également nécessaires sur la conception du système. De plus, les entrées du système sont difficiles à recréer. Sur un prototype, aucune hypothèse sur la conception du système et son environnement n'a besoin d'être faite par définition. Les injections matérielle et logicielle ont toutes deux aussi des inconvénients.

L'injection matérielle est lourde en coût de déploiement et l'injection logicielle, moins coûteuse, souffre d'un accès restreint à la totalité du système.

Nous donnons dans ce paragraphe quelques exemples d'outils d'injection de fautes. MESSALINE [Arlat 1990] est un outil d'injection de fautes physiques pour des prototypes de système à valider. Dans [Tan 2012] les auteurs injectent des fuites mémoires (i.e. mémoire non libérée par une application), des boucles d'exécution infinies, et des charges de travail importantes menant à des goulots d'étranglement pour les applications. Cloudval [Pham 2011] est un framework d'injection par des techniques logicielles (en anglais SoftWare-Implemented Fault Injection ou SWIFI). Il permet de positionner des points d'arrêt dans le code d'une application cible et de

modifier un objet donné par bitflip, modification d'adresse ou d'instruction. Vajra [Wierman 2008] est un framework d'injection de fautes dans le but d'étalonnage. Il permet d'injecter dans un système des pertes de messages, des corruptions de messages, des latences dans l'envoi de messages, des arrêts de fonction d'une application, des gels de fonction d'une application et des fuites de mémoire. Les injections sont réalisées à partir de l'interception d'appels à des librairies partagées en C. NFTAPE [Stott 2000] est un environnement d'injection de fautes configurable supportant les systèmes distribués. NFTAPE supporte l'utilisation de plusieurs types d'injecteurs de faute (logiciels ou matériels) et permet de tester complètement un système avec une grande couverture de fautes.

2.2.2 Métriques d'évaluation

La détection d'anomalies s'évalue par l'analyse des anomalies ayant été détectées à tort ou à raison comme des anomalies ou comme des comportements normaux.

Les approches qui nous intéressent pour la détection d'anomalies sont la classification ainsi que le clustering. La classification en particulier s'évalue en vérifiant a posteriori si les classes affectées aux observations sont les bonnes. Le clustering quant à lui n'est pas développé pour affecter une signification sémantique aux clusters identifiés (cluster d'anomalie ou non). Deux cas de figures se posent donc pour l'évaluation de performance du clustering. Le premier cas est celui pour lequel les observations qui ne sont pas affectées à des clusters sont considérées comme des anomalies. Le deuxième cas consiste à utiliser une procédure de classification après le traitement des clusters afin d'identifier si un cluster représente ou non une anomalie.

Dans le domaine de l'apprentissage automatique, une procédure de classification apprend et prédit des labels dits positifs ou négatifs. Les comportements qui sont qualifiés dans nos travaux de *Négatifs* sont les comportements normaux et de *Positifs* pour les cas d'anomalie. Les résultats attendus et les résultats obtenus par la détection (à apprentissage supervisé ou non supervisé) sont alors organisés dans une matrice de confusion (voir le tableau 2.1) composées des métriques principales suivantes :

- Vrai négatif ou True Negative (TN) : comportement normal prédit comme tel.
- Vrai positif ou True Positive (TP) : anomalie prédite comme telle.
- Faux positif ou False Positive (FP) : comportement normal prédit comme une anomalie.
- Faux négatif ou False Negative (FN) : anomalie prédite comme comportement normal.
- Positifs ou Positives : nombre total d'anomalies ($P = TP + FN$)
- Niégatifs ou Negatives : nombre total de comportements normaux ($N = TN + FP$)

Plusieurs métriques peuvent être déduites de ces métriques principales. Parmi celles qui sont utilisées dans notre domaine nous trouvons l'*exactitude*, la *précision*, le *rappel* aussi appelé taux de vrais positifs (TVP) et le taux de faux positifs (TFP).

L'exactitude correspond à la proportion d'observations correctement prédites. La précision représente la probabilité qu'une observation classée Positif soit effectivement Positif. Elle correspond donc au nombre total de Vrai Positifs renvoyés, sur le nombre total d'observations ayant été renvoyées comme Positifs, à tord ou à raison. Le rappel quant à lui représente le taux de Vrais Positifs. Il correspond donc au nombre total de Vrai Positifs renvoyés, divisé par le nombre total de Positifs que l'approche aurait dû reconnaître. La définition de ces mesures est donnée dans le tableau 2.2

	Positifs Réels	**Négatifs Réels**
Positifs Prédits	TP	FP
Négatifs Prédits	FN	TN

Tableau 2.1 – Matrice de confusion.

Mesure	**Formule**
Exactitude	$\frac{TP+TN}{TP+TN+FP+FN}$
Précision	$\frac{TP}{TP+FP}$
Rappel (ou TVP)	$\frac{TP}{TP+FN}$
TFP	$\frac{FP}{FP+TN}$

Tableau 2.2 – Définition des mesures de performance.

Ces mesures sont notamment utilisées dans les travaux de détection présentés dans la section précédente [Zhang 2013, Tan 2012, Wang 2010, Nguyen 2013, Xu 2009, Farshchi 2015a].

Les courbes "efficacité du récepteur" (en anglais Receiver Operating Characteristic ou *ROC*) [Provost 1998] et précision-rappel (PR) sont souvent utilisées pour résumer ces métriques pour plusieurs seuils de prédiction (par exemple dans [Dean 2012, Guan 2012b, Miyazawa 2015]). Une courbe ROC présente le TVP en fonction du TFP obtenus pour plusieurs seuils de prédiction (2.1.3). Une classification à prédiction parfaite atteint un TFP de 0 et un TVP de 1 pour chaque seuil de prédiction (représenté en rouge sur la figure 2.2.1 (a)). Une classification aléatoire amène à une courbe ROC représentant une droite entre les points $(0, 0)$ et $(1, 1)$ (représenté en bleu sur la figure 2.2.1 (a)).

Les courbes ROC s'expriment en fonction de taux. Elles sont donc indépendantes de la distribution des classes des données (i.e. la proportion d'observations associées à chacune des classes). Les résultats obtenus par analyses de telles courbes sont donc potentiellement généralisables à des jeux de données possédant une distribution différente (possédant moins d'anomalies par exemple) [Davis 2006]. L'aire en dessous de la courbe ROC (en anglais Area Under the ROC Curve ou ROC

(a) Courbes ROC. (b) Courbes PR.

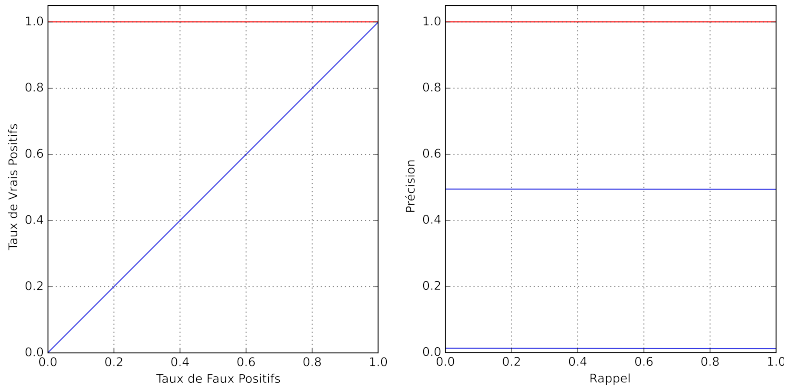


FIGURE 2.2.1 – Courbes ROC et PR pour une classification parfaite (en rouge) et une classification aléatoire (en bleu).

AUC) constitue une métrique qui généralise la performance d'un modèle de détection [Bradley 1997]. L'AUC est en effet comme la courbe ROC indépendante des changements de distribution, et elle est de plus indépendante au seuil de prédiction.

Une courbe PR présente la précision en fonction du rappel qui sont obtenus pour plusieurs seuils de prédiction. La précision est une mesure qui varie grandement avec la distribution des classes dans un jeu de données. Une classification à prédiction parfaite est représentée en rouge sur la figure 2.2.1 (b) et peut atteindre un rappel et une précision de 1. Une classification aléatoire amène à une courbe PR représentant une droite appelée en anglais *baseline* d'équation $y = P/(P+N)$ (représentée en bleu sur la figure 2.2.1 (b) pour $y = 2\%$ et en violet pour $y = 50\%$). Plus la proportion de représentants négatifs est petite, plus la précision est sensible au moindre FP et la baseline est basse sur une figure PR. Nous prenons ici l'exemple d'un jeu de données de 100000 observations de comportement normal (i.e. $TN + FP = 100000$) et 100 observations d'anomalie (i.e. $TP + FN = 100$). Le jeu de données comprend environ 1% d'anomalie. Il est une représentation schématique d'un système réel avec peu d'anomalie. Dans le cas où un fournisseur accepte 5% de FP (i.e. $FP = 505$) et impose au moins 90% de TP (i.e. $TP = 90$), nous obtenons sans surprise un rappel excellent de 0.9 mais une précision de 0.15. La précision obtenue est non satisfaisante a priori, toutefois l'hypothétique modèle prédit selon les exigences du fournisseur. Pour des distributions similaires, la précision ne pourra jamais être satisfaisante.

En conséquence, les courbes PR sont variables selon la distribution évoquée et il est important de préciser cette distribution lors de la présentation de telles courbes.

Il est à noter que le cas d'une classification multi-classe doit être traité d'une manière particulière pour évaluer les observations correctement prédites selon leur classe. L'évaluation multi-classe peut se traiter par la décomposition de la classifi-

cation en plusieurs problèmes binaires. Dans le cas le moins complexe on crée un classifieur pour chaque classe donnée permettant de faire des prédictions sur cette classe contre toutes les autres classes. On parle alors de classification "un contre tous". La décision quant à la présence d'une anomalie est donnée par le classifieur qui présente la plus forte probabilité de détection d'un positif. Il est possible aussi de créer un classifieur pour chaque paire de classes donnée. Ces méthodes sont comparées notamment dans [Hsu 2002].

Il existe donc plusieurs mesures pour évaluer les performances d'une technique de détection d'anomalies. Selon son contexte de travail et ses objectifs, c'est à l'utilisateur de choisir quelle mesure est à considérer en priorité et à optimiser. Tout dépend de considérations telles que présentées dans 2.1.3.3 par exemple.

L'évaluation est donc une tâche qui doit être réalisée avec méthode et de manière rigoureuse. Lorsque cela n'est pas le cas, les tests en cas réels donnent souvent de mauvais résultats en comparaison avec les résultats de validation, comme le montre [Allix 2016].

2.3 Conclusion

Dans ce chapitre nous avons présenté plusieurs travaux dédiés à la détection d'anomalies dans de grands systèmes informatiques en général et liés au cloud computing en particulier. Les approches à cette problématique sont nombreuses et plus ou moins complexes selon le nombre d'étapes à prendre en compte, le temps de paramétrage, et le nombre de modèles à entrainer. Certaines de ces approches s'appliquent en ligne mais ne sont pas adaptées à des changements de charge de travail par exemple. D'autres sont complexes et ne permettent pas de réagir rapidement lors de la détection d'une anomalie. Nos critères de sélection d'une technique de détection d'anomalies pour notre contexte de services cloud ne sont donc souvent pas tous respectés.

Il est de plus souvent difficile d'évaluer dans quelles mesures les tests de validation d'une technique de détection par apprentissage automatique attestent des performances de la technique dans un contexte industriel, voire plusieurs contextes industriels. Cela est souvent dû à des tests incomplets ou biaisés de par les méthodes de validation utilisées et leur configuration parfois hasardeuse.

Dans nos travaux, nous proposons une stratégie de détection d'anomalies qui répond à la problématique présentée dans le chapitre 1. Cette stratégie repose sur des techniques d'apprentissage automatique exploitant les données de performance système collectées en ligne depuis les services cloud déployés par un fournisseur. Par le fait qu'elle repose sur l'analyse de données de performance système, cette stratégie peut être appliquée indépendamment du service étudié. Nous montrons dans ce manuscrit que son implémentation est de plus très peu dépendante de ce dernier. Elle repose également sur une technique d'injection de fautes permettant d'introduire des anomalies dans un système cible afin d'entrainer les modèles d'apprentissage automatique à discriminer les comportements anormaux des comportements nor-

maux.

Notre stratégie repose en dernier lieu sur deux entités dont l'action peut être menée de différentes manières. L'entité de collecte de données de performance système travaille avec deux sources différentes. L'une collecte des données à partir des systèmes d'exploitation hôte du service considéré et l'autre collecte des données grâce aux hyperviseurs déployés dans le cloud qui hébergent ce service. L'entité de détection fonctionne avec des niveaux de détection d'anomalies différents, à savoir la détection d'erreur, de symptômes préliminaires à une violation de service, et de violation de service. Plusieurs types de détection peuvent également être mis en œuvre selon si un diagnostic est réalisé quant à l'origine supposée d'une anomalie en cours.

Grâce à l'évaluation de cette stratégie sur deux cas d'étude, nous mettons en évidence des tests à mettre en œuvre qui sont nécessaires à la collecte méthodique d'un jeu de données d'entrainement complet et dans un temps restreint. Nous mettons de plus en évidence les tests pour une validation rigoureuse d'une technique de détection d'anomalies. La durée de validité des modèles de détection obtenus est également évaluée avec une méthode comparable à celle de [Scandariato 2014]. L'analyse des compteurs de performance de monitoring utiles à la détection d'anomalies lors de ces tests est également faite. Elle nous permet de comprendre de quelle manière les erreurs sont détectées et donc de quelle manière elles se manifestent.

Nous étudions pour cela des techniques simples et n'accumulant ainsi pas les défauts de performance que l'association d'un nombre trop important de modèles de détection pourrait entrainer. La première technique repose sur l'apprentissage supervisé uniquement. La deuxième technique est une nouvelle technique de détection DESC, exploitant la capacité des algorithmes de clustering à trouver des structures dans les données. Cette technique est fondée sur une caractérisation de comportement utilisant le barycentre de clusters de données. Enfin nous évaluons dans quelle mesure un diagnostic sur l'origine d'une anomalie peut être fait en même temps que la détection de l'anomalie.

Leurs performances de détection sont évaluées en présence d'anomalies grâce à la mise en œuvre d'une plateforme d'expérimentation. Nous avons déployé trois cas d'étude sur cette plateforme dont deux sont présentés dans cette thèse (le troisième cas d'étude est présenté et étudié dans nos précédents travaux [Silvestre 2015b]).

Chapitre 3

Présentation générale de la stratégie de détection

Sommaire

3.1 Cadre conceptuel **37**
3.1.1 Contexte 38
3.1.2 Hypothèse sur la manifestation d'une anomalie 38
3.2 Entité de monitoring **39**
3.2.1 Différentes sources de monitoring 39
3.2.2 Groupement des compteurs de performance 40
3.3 Entité de détection **42**
3.3.1 Niveau de détection 42
3.3.2 Type de détection 44
3.3.3 Détection par apprentissage supervisé 44
3.3.4 Détection DESC 46
3.4 Conclusion **57**

L'objectif de ce chapitre est de présenter la stratégie de détection que nous avons définie. Pour ce faire, nous commençons par présenter le cadre conceptuel ainsi que l'hypothèse qui sous-tend cette stratégie. Nous présentons ensuite l'entité de collecte de données de performance système, que nous appelons entité de monitoring, avant d'aller plus en détail vers dans la description de l'entité de détection contenant les techniques de détection d'anomalies proprement dites.

3.1 Cadre conceptuel

Cette section présente le cadre conceptuel de nos travaux. Elle définit en premier lieu notre contexte d'étude et en deuxième lieu nos hypothèses de travail.

3.1.1 Contexte

Nous considérons l'analyse des services cloud du point de vue d'un fournisseur proposant divers services cloud. Notre objectif général est de permettre à un tel fournisseur d'assurer la détection d'anomalies dans ses services cloud tout en répondant à la problématique présentée dans le chapitre 1. La détection doit être automatique, se réaliser en ligne, s'adapter aux changements de charge de travail et ne pas être dépendante de la fonction ou de l'implémentation des services du fournisseur.

Notre stratégie considère comme *système cible* d'une détection d'anomalies (appelé système dans la suite de cette thèse), un service cloud. Le système est potentiellement réparti sur plusieurs VMs.

Considérer chaque service un à un est nécessaire. Un ensemble de services contribuant à une même fonction ou à des fonctions différentes, est trop hétérogène pour pouvoir agréger les données en une information intelligible.

3.1.2 Hypothèse sur la manifestation d'une anomalie

Comme nous l'avons vu dans le chapitre 1, la sélection d'une technique pour la détection d'anomalies est notamment faite en fonction des données analysées. Dans le contexte de la détection dans des systèmes informatiques, les hypothèses faites sont liées à la nature du système étudié ainsi qu'à la collecte des données de ce système. Nous choisissons dans nos travaux d'appliquer la détection d'anomalies sur les données de monitoring d'un système cible.

Dans ce cas, une anomalie peut se manifester par un évènement ponctuel associé au dépassement d'un seuil de valeur, ou autrement dit une variation importante, pour une métrique, ou bien par une fréquence de variation inattendue dans les fluctuations d'une métrique, ou encore par une valeur constante d'une métrique normalement variante. Dans nos travaux nous considérons uniquement la détection d'évènements ponctuels menant à des variations importantes des valeurs d'une métrique.

Une variation importante peut effectivement avoir lieu dans trois situations : 1) une modification de configuration, 2) un pic de charge de travail ou bien 3) une anomalie survenue dans une ressource système ou dans une application. Dans ces trois situations, une alarme levée doit être traitée d'une manière différente. Dans le premier cas, les alarmes levées doivent être évitées en entrainant de nouveau les modèles de détection ou bien en ne prenant pas en compte les alarmes apparues directement après la variation importante. Dans le deuxième cas, une alarme est levée si aucune mise à l'échelle des ressources n'a été réalisées pour gérer le pic de charge (il aurait en effet pu être prédit par des algorithmes de prédiction de besoin en ressources ou de reconfiguration comme proposé dans [Cerf 2016]). Dans le troisième cas, une alarme est levée et un diagnostic doit être fait pour retirer l'anomalie en cours.

Finalement, nous émettons l'hypothèse suivante :

Hypothèse 1 (H1): *Une anomalie est un changement de comportement et se traduit par une variation importante de certaines métriques décrivant un système.*

3.2 Entité de monitoring

La surveillance de système informatique vise à vérifier l'état du système de manière continue et par plusieurs moyens d'observation possibles. Ces moyens sont souvent complémentaires (logs, traces d'audit, données de performances système, voir 2.1).

Dans nos travaux nous considérons les définitions suivantes. La surveillance d'un système permet d'obtenir plusieurs indicateurs de performance de ce système que nous appelons *compteurs de performance*. Les compteurs de performances qualifient les ressources d'un système comme l'activité CPU ou bien la rapidité à laquelle une interface réseau réceptionne des trames réseau. Périodiquement, le système de surveillance relève les valeurs, ou métriques, des compteurs et les groupe en vecteurs appelés *observations*. C'est donc sur ces observations que la détection d'anomalies se repose et elle est d'autant plus complexe que le nombre d'attributs (i.e. le nombre de dimensions) d'une observation est important.

Nos travaux portent sur l'analyse des compteurs de performance système et réseau qui sont indépendants de la fonction du service. Cette surveillance de système est souvent désignée par sa traduction en anglais à savoir, le *monitoring de système* ou *monitoring*.

Dans un système virtualisé, le monitoring d'un système peut se faire par deux moyens, soit au niveau des VMs, soit au niveau de l'hyperviseur comme nous allons le présenter dans la suite.

3.2.1 Différentes sources de monitoring

Nous rappelons que notre système cible est potentiellement réparti sur plusieurs VMs. Ici, une VM constitue la brique logique de notre étude de système. Le monitoring d'un système cible correspond au monitoring de chacune des VMs qui le composent.

Deux sources de monitoring sont considérées dans nos travaux à savoir les SEs des VMs du système cible, et les hyperviseurs qui hébergent ces VMs. Elles sont représentées sur la figure 3.2.1 et détaillées ci-dessous.

— Les données de monitoring d'un système peuvent être obtenues grâce à des compteurs de performance des SEs sur lesquels sont installés chacune des applications du système cible. Ce type de monitoring nécessite l'installation d'un agent de monitoring dans les SEs respectifs des applications. Cette source de monitoring est appelée *monitoring SE*. Elle suppose de connaitre les différents SEs utilisés dans le système cible ainsi que la possibilité d'installation d'un logiciel tiers. Elle permet d'obtenir de nombreux compteurs de performance liés à l'usage détaillé des ressources de la VM par le SE.

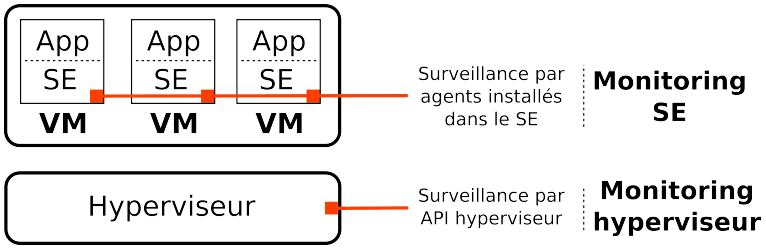


FIGURE 3.2.1 – Sources de monitoring.

— Le monitoring d'un système cible virtualisé peut aussi se faire grâce aux hyperviseurs hébergeant les VMs qui composent le système cible. En effet, un hyperviseur peut avoir connaissance de plusieurs compteurs de performance des VMs qu'il ordonnance et auxquelles il alloue des ressources. Les données sont alors collectées par le biais de l'interface de programmation (en anglais Application Programming Interface ou API) des hyperviseurs. Cette source de monitoring est appelée *monitoring hyperviseur*. Le nombre de compteurs disponibles est souvent moins important que celui proposé par un monitoring SE. Cela est dû à la vue partielle qu'à un hyperviseur sur un système qu'il héberge (il n'a pas la vue fonctionnelle du système). Elle ne demande aucune installation particulière dans les VMs ou dans les hyperviseurs.

Aucune connaissance de la fonction et des spécifications des applications déployées dans les VMs n'est requise pour la mise en place de ces sources de monitoring.

3.2.2 Groupement des compteurs de performance

Les compteurs de performance d'un système cible sont groupés périodiquement afin de constituer des observations. Les observations sont par la suite fournies à l'entité de détection.

Ce groupement peut être faite de deux manières différentes. Les compteurs peuvent être groupés selon s'il représentent une même VM ou selon s'ils décrivent une même ressource d'une même VM (par exemple : CPU de VM_A, mémoire de VM_A, CPU de VM_B etc.).

Lorsqu'un test de détection d'anomalies est réalisée à partir des compteurs de monitoring de VM_A ou bien des compteurs de monitoring de ses ressources (CPU de VM_A etc.), on dit que VM_A est la *VM observée*. C'est l'observation de VM_A qui permet d'obtenir les performances de détection lors du test.

3.2.2.1 Groupement par-VM

Le *groupement par-VM* consiste à grouper les compteurs en fonction des VMs du système cible qu'ils décrivent. Les observations relatives à une seule VM sont donc traitées séparément (i.e. par des modèles différents) des observations des autres

VMs. Cela permet une parallélisation des calculs nécessaires à la création de modèles ainsi qu'à la prédiction. Cela permet aussi de représenter par différents modèles le comportement de VMs d'un même système cible qui peuvent avoir des comportements très différents (qu'ils subissent une défaillance ou non). La figure 3.2.2 représente le groupement par-VM des compteurs de performance d'un système de N VMs.

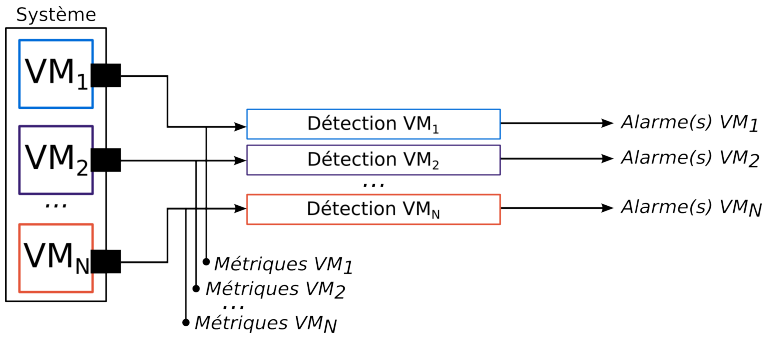


FIGURE 3.2.2 – Groupement des compteurs par-VM.

3.2.2.2 Groupement par-ressource

Le *groupement par-ressource* consiste à grouper les compteurs en fonction des VMs (de façon similaire au groupement par-VM) ainsi qu'en fonction des ressources de ces VMs. Les catégories de ressources considérées dans nos travaux sont la CPU, la mémoire, le disque ainsi que le réseau (d'autres catégories plus précises pourraient être définies).

Ce groupement, comme le groupement par-VM, permet une parallélisation des calculs nécessaires à la création de modèles ainsi qu'à la prédiction. La figure 3.2.3 représente le groupement par-ressource des compteurs de performance d'un système de N VMs avec 3 catégories de compteurs : CPU, mémoire et disque.

3.2.2.3 Discussion sur le diagnostic

Il est à noter qu'une anomalie peut être détectée dans des données dont les attributs ne la qualifient pas a priori directement. Par exemple, une surchauffe d'un serveur peut être détectée par des compteurs de rotations de ventilateurs. Dans ce cas, la corrélation se trouve entre la température d'un serveur et le nombre de rotations par minute de ses ventilateurs. Les sondes surveillant les ventilateurs peuvent donc permettre la détection d'anomalies affectant la température d'un système.

Une anomalie peut aussi être détectée à partir d'une ressource qui n'est pas la cause du comportement détecté. En voici deux exemples.

— Si l'on considère un disque défaillant avec un temps d'accès trop important, il s'avère que la CPU du système sur lequel est monté le disque peut certainement se trouver en sous activité. En effet, les accès disque peuvent mettre

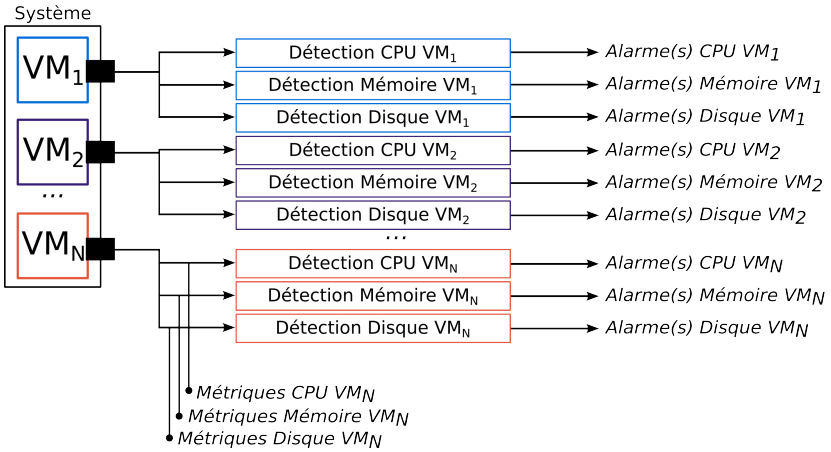


FIGURE 3.2.3 – Groupement des compteurs par-ressource.

en attente la CPU. Dans ce cas, la CPU peut être détectée comme ayant un comportement anormal, tout comme le disque.

— Dans le cas d'un serveur informatique, une erreur d'un périphérique peut provoquer une erreur dans une application utilisant ce périphérique. Dans ce cas, l'erreur s'est propagée, causant d'autres erreurs dans le périphérique jusqu'à atteindre l'interface entre l'application et le périphérique.

Une alarme levée suite à une détection faite sur les compteurs de performances liés à une seule ressource d'une VM ne permet donc pas nécessairement de conclure à une anomalie ayant cette ressource pour origine. Un outil de diagnostic avancé doit être employé afin de corréler et analyser l'occurrence dans le temps des anomalies détectés grâce aux différents groupements de compteurs.

3.3 Entité de détection

3.3.1 Niveau de détection

Dans notre contexte, une défaillance est définie par la déviation du service délivré de la *spécification* du système. Cette dernière correspond à une description agréée du service attendu du système cible. La spécification du service attendu peut être exprimée sous différentes formes. Elle peut par exemple être exprimée par des attributs de la sûreté de fonctionnement à garantir, ou bien en termes de mesures à respecter (mesure du temps de réponse, de la bande passante, etc.) [ETSI 2012].

D'un point de vue fournisseur, une défaillance est définie en fonction d'une spécification exprimée dans un contrat liant le fournisseur et un utilisateur. Plusieurs spécifications système haut niveau sont souvent étudiées dans les services cloud et les plus populaires sont la rapidité d'accès à un grand nombre de données du service (en termes de latence, et de débit) et la disponibilité du service [ETSI 2012, Serrano 2016].

Dans nos travaux, nous considérons qu'un service est spécifié par le pourcentage de requêtes utilisateur traitées avec succès par minute par le système (appelé PS pour pourcentage de succès). Nous utilisons particulièrement le pourcentage de requêtes utilisateur non traitées avec succès par minute afin d'évaluer si un système délivre un service correct (appelé PE pour pourcentage d'échec, avec $PE = 1 - PS$).

Un service est délivré correctement tant que PE ne dépasse pas un certain seuil *PE_max*. Lors du dépassement de ce seuil, une violation de service à lieu. La violation de service correspond à la défaillance. Nous utilisons le terme de *violation de service* afin de garder en tête qu'il s'agit d'une défaillance par rapport à la définition d'un service qui est fonction de *PE_max*.

Un symptôme préliminaire à une violation de service correspond à l'état décrivant le comportement d'un système lors d'une période $[t - \delta_t, t]$ où t est l'instant d'occurrence de la violation de service. La figure 3.3.1 représente les valeurs de PE en fonction du temps pour un système donné lors d'une violation de service. Les périodes de symptômes préliminaires et de violation de service y sont représentées.

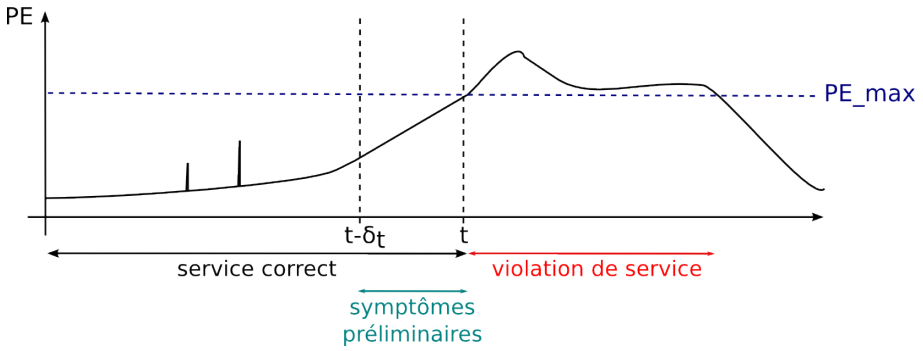


FIGURE 3.3.1 – Représentation de PE pour un service correct, des symptômes préliminaires à une violation de service et d'une violation de service.

Il est donc important pour un fournisseur de détecter les symptômes à des violations de service ainsi que les violations de service elles-mêmes (dans le cas où les symptômes n'ont pas été détectés). Il est de plus important de pouvoir détecter des erreurs. Les erreurs sont en effet l'activation de fautes qui donnent potentiellement lieu à des symptômes puis des violations de service. Dans certains cas, une erreurs peut être transitoire et ne jamais affecter le service rendu à un client.

Il est donc possible de distinguer trois genres d'anomalies et donc de mener *trois niveaux de détection d'anomalies* :

1. détection d'erreur,
2. détection de symptômes préliminaires à une violation de service,
3. détection de violations de service.

Dans nos travaux, nous effectuons la détection sur ces trois niveaux à partir des données de monitoring. De cette manière, nous n'avons à gérer qu'un seul type de données et qu'une seule entité de collecte de données.

Ces trois niveaux de détection sont intégrés à notre stratégie de détection et doivent être actifs en permanence. Ainsi, lorsqu'une détection d'erreurs n'a pas été faite dans le système, la détection de symptômes préliminaires à la violation de service liée à cette erreur peut tout de même lever une alarme à l'attention des administrateurs cloud. Si les symptômes ne sont pas bien détectés, la détection de violations de service est la dernière détection permettant d'informer un administrateur d'un besoin de modification de la configuration du service.

3.3.2 Type de détection

Dans nos travaux nous évaluons les performances de détection de notre stratégie pour les trois niveaux de détection présentés plus haut.

Cette évaluation est de plus caractérisée par un double but : 1) la détection en elle même d'une anomalie, et 2) effectuer un diagnostic sur l'origine de l'anomalie détectée lors d'une détection d'anomalies.

Lorsqu'aucun diagnostic n'est fait, la détection est de type *binaire* (car elle est mise en œuvre par un algorithme de classification binaire) et consiste à statuer si oui ou non il existe une anomalie dans la VM observée.

Lorsqu'un diagnostic est fait, la détection d'anomalies (peut importe le niveau de détection considéré) est multi-classe (car elle est mise en œuvre par un algorithme de classification multi-classe). Dans nos travaux, un diagnostic est réalisé selon deux aspects et amène nos deux derniers types de détection :

1. *le type d'erreur à l'origine de l'anomalie*,
2. *la VM qui est à l'origine de l'anomalie.*

Il est à noter qu'une anomalie peut être détectée en même temps que d'autres anomalies.

— Dans le cas où un symptôme préliminaire à une violation de service ou une violation de service est détecté en même temps qu'une erreur, la cause adjugée peut être celle pointée par la détection d'erreur.
— Dans le cas où aucune erreur n'est détectée durant la détection d'un symptôme préliminaire à une violation de service ou d'une violation de service, il est intéressant d'analyser le diagnostic fait par la détection en cours s'il existe.
— Dans le cas où des diagnostics différents sont fournis par des erreurs détectées simultanément, il est nécessaire de procéder à un vote des différents modèles de détection et opter pour le diagnostic ayant le plus de vote.

L'étude de la corrélation entre des anomalies simultanées est toutefois hors du cadre de nos travaux.

3.3.3 Détection par apprentissage supervisé

3.3.3.1 Fonctionnement en deux phases

Un algorithme à apprentissage supervisé, et donc une technique de détection par apprentissage supervisé, s'exécute en deux phases, à savoir une phase d'entraine-

ment et une phase de détection. La première est exécutée hors ligne durant un temps prédéfini et la seconde en ligne sur une longue période. Ces phases d'exécution sont représentées dans les figures 3.3.2 (1) et (b) pour la détection à partir des données d'une VM du système cible représenté dans 3.2.2. Durant la phase d'entrainement, les observations du système sont collectées et stockées en base de données. Des injections de différents types dans le système permettent de collecter des observations de comportement normal et anormal. Les observations sont par la suite traitées hors ligne par centrage/réduction et labelisées. Les labels sont positionnés en fonction de notre connaissance des instants d'injection). Le traitement fini, les observations sont regroupées dans un jeu de données qui est donné en entrée de la création de modèles. Durant la phase de détection, les observations du système sont directement traitées en ligne par centrage/réduction et fournies aux modèles. Ces derniers effectuent leurs classifications sur chaque réception d'une nouvelle observation, dont le résultat est traduit par une potentielle alarme.

(a) Phase d'entrainement, hors ligne.



(b) Phase de détection, exécution périodique en ligne.

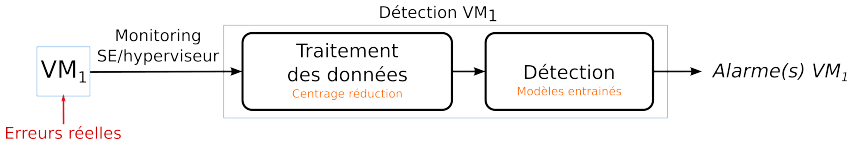


FIGURE 3.3.2 – Deux phases d'exécution de la technique de détection par apprentissage supervisé.

Dans nos précédents travaux [Silvestre 2014] nous avons étudié la faisabilité d'une détection binaire à partir des données de monitoring d'un système correspondant à nos deux cas d'étude. Dans ce manuscrit, nous étudions dans quelles mesures la classification supervisée parvient à traiter les différents types de détection qui ont été présentés dans la sous section 3.3.2 et notamment la détection multi-classe permettant un diagnostique d'anomalie. Nous mettons en évidence des tests à mettre en œuvre qui sont nécessaires à la collecte méthodique d'un jeu de données d'entrainement complet et dans un temps restreint. Nous mettons de plus en évidence des tests pour la validation rigoureuse d'une technique de détection d'anomalies.

Enfin, nous utilisons plusieurs algorithmes populaires dans le domaine de l'apprentissage automatique pour mettre en œuvre notre technique de détection afin de comparer leurs performances, à savoir : un algorithme fondé sur le théorème de Bayes [Chan 1982], l'algorithme du plus proche voisin (en anglais Nearest Neighbor) [Beyer 1999], un réseau de neurones supervisé [Gardner 1998]

et l'algorithme Random Forest [Breiman 2001]. Les algorithmes de SVM, Random Forest et Gradient Boosting on été étudiés dans nos précédents travaux [Silvestre 2014, Silvestre 2015b]. Les performances de détection obtenues dans ces travaux sont excellents pour les trois algorithmes. L'algorithme Random Forest s'avère donner les meilleures performances de détection avec les plus petits temps de prédiction. Pour cette raison, nous présentons dans ce manuscrit la comparaison de l'algorithme Random Forest avec d'autres algorithmes. L'algorithme SVM quand à lui est l'algorithme au temps d'execution le plus lent.

3.3.3.2 Groupement des données

Pour cette technique de détection nous choisissons le groupement par-VM. En effet, les algorithmes que nous testons dans nos travaux peuvent supporter l'étude de jeux de données avec un grand nombre d'attributs (i.e. de dimensions) donc le groupement par-VM est possible. Nous ne choisissons pas le groupement par-ressource car elle nécessite de traiter un nombre plus important de modèles (pour chaque ressource de chaque VM).

3.3.4 Détection DESC

Des changements de charge de travail, les mises à jour ou les reconfigurations de certaines ressources peuvent survenir dans lors de l'éxécution d'un service. Ces changements peuvent amener des modèles supervisés à être obsolètes. Ce problème est dû à l'incomplétude du jeu de données d'entrainement de ces modèles qui ne peut pas prendre en compte tous les contextes d'utilisation d'un service mais au mieux les plus caractéristiques. Ainsi, une technique de détection seulement fondée sur un algorithme à apprentissage supervisé pourrait avoir des difficultés à s'adapter à l'évolution d'un système.

Ce point nous a motivé à définir une technique de détection en deux phases. La première phase repose sur la caractérisation du comportement d'un système de manière non supervisée et en ligne. Contrairement aux travaux présentés dans le chapitre 2, cette technique s'émancipe du seuil qui détermine à partir de quelle distance un cluster est trop éloigné de la majorité des clusters et devient un cluster aberrant (i.e. qui traduit de la présence d'une anomalie dans les données traitées, en anglais *outlier*). Elle comprend une étape de clustering en ligne de données de monitoring puis une étape de calcul de barycentre des clusters. Ce sont notamment les déplacements du barycentre des clusters qui amèneront une anomalie a être détectée. Ce barycentre permet de calculer des descripteurs simples caractérisant le modèle de clusters à un instant t. La seconde phase correspond à une analyse des comportements représentés durant la première phase. En effet, la variation des valeurs des descripteurs dans le temps nous permet d'évaluer si une anomalie est présente ou non dans le système cible. Nous appelons cette technique DESC (pour descripteurs).

Dans la suite nous donnons plus de détail sur les évaluations préliminaires et la

description de notre technique. L'évalution principale de celle-ci est faite dans les chapitres 5 et 6.

3.3.4.1 Analyse des données de monitoring et hypothèses sur les clusters

Une expérimentation préliminaire a été réalisée afin de sonder le comportement d'un système en présence d'anomalies. Cette expérimentation porte sur des données collectées à partir de notre premier cas d'étude MongoDB que nous présentons dans le chapitre 5. Lors de cette expérimentation, nous avons injecté des anomalies dans une VM d'un déploiement de base de données MongoDB [Mon 2016]. Une charge de travail constante active les fonctions de la base de données en lecture seulement. Nous avons collecté 10000 observations de monitoring de cette VM.

Les données ont été traitées avec le groupement par-ressource présentée dans 3.2.2.2. Les observations d'une VM correspondant à une ressource R font partie du *sous-flux* de données R. Chaque sous-flux de données est donc traité séparément. Lors de cette expérimentation nous avons d'abord analysé les métriques du jeu de données pour ensuite appliquer un algorithme de clustering sur ces métriques et constater des propriétés liées à la création des clusters.

1. Dans un premier temps nous avons analysé les distances entre les observations du jeu de données. Nous avons remarqué que pour un sous-flux donné, une anomalie injectée dont le type est liée aux compteurs de performance du sous-flux (par exemple une anomalie monopolisant l'activité CPU si l'on considère le sous-flux CPU) donne lieu à des observations qui sont éloignées des observations de comportement normal enregistrées juste avant l'anomalie . L'éloignement est constaté en moyenne et avec une distance euclidienne. Cet éloignement est calculé sur les observations de comportement normal enregistrées juste avant l'anomalie. Ce sont en effet ces observations qui représentent le comportement du système à partir duquel nous souhaitons détecter une déviation de comportement. Cette constatation sur les distances entre observations nous a conduit à penser qu'il en serait de même si les données étaient mises dans des clusters. Autrement dit, une anomalie injectée pourrait donner lieu à la création de clusters dans le modèle de clusters de certains sous-flux. Ces clusters pourraient être éloignés des clusters de comportement normal représentant le système cible.
2. Dans un deuxième temps, nous avons appliqué sur les données un algorithme de clustering de flux de données appelé rEMM [Hahsler 2010]. rEMM traite les observations qu'il reçoit une par une dans le sens chronologique. Il crée des clusters qui ont un poids proportionnel au nombre d'observations qui ont été affectées au cluster. Le poids des clusters est diminué à chaque itération de l'algorithme (une itération correspond au traitement d'une nouvelle observation) afin de représenter un vieillissement de cluster. Ce point est particulièrement intéressant dans le cadre de nos travaux. Il est en effet pertinent de considérer que des clusters représentant un système à un instant t

ne le représentent plus à l'instant $t + n$, car le système évolue et sa charge de travail est susceptible de changer. L'algorithme attribue à un cluster qui ne s'est pas vu affecter de nouvelles observations durant un certain nombre d'itérations, un poids proche de 0. Il est possible d'élaguer du modèle, les clusters dont le poids est inférieur à une certaine limite. L'algorithme permet de considérer les clusters créés comme des états d'une chaine de Markov et d'en analyser les transitions.

L'analyse de ces transitions ne nous a pas permis de conclure à l'utilité des chaines de Markov pour nos données. Nous avons toutefois analysé les clusters créés lors d'injections. Notre technique est algorithmiquement simple car elle nous utilisons seulement la composante de création de clusters de rEMM. L'algorithme était alors configuré avec les paramètres par défaut (notamment en distance euclidienne) mais avec un facteur de vieillissement des clusters non nul. Pour chaque observation collectée, nous avons cherché si une fois traitée par l'algorithme de clustering, celle-ci donnait lieu à la création d'un cluster ou si celle-ci avait été affectée à un ancien cluster. Nous avons constaté expérimentalement qu'*une anomalie implique la création d'un cluster* dans le modèle de clusters de certains sous-flux de données. Toutefois des clusters peuvent aussi être créés lors du traitement d'observations de comportement normal.

Nous avons également testé l'algorithme de clustering en ligne Clustream [Aggarwal 2003] repose sur la construction de micro clusters qui sont assemblés entre eux au cours du temps. Cependant la configuration de cet algorithme s'est avérée fastidieuse. Il possède en effet un nombre de paramètres important pour notre cas d'étude et l'étude de sensibilité de chaque paramètre s'est avérée trop longue compte tenu de notre contexte de déploiement pour service cloud. L'algorithme Denstream [Cao 2006] constituant des clusters selon la densité de données dans l'espace s'est avéré lui aussi long à configurer. De plus il n'a pas permis de constater que des anomalies pouvaient correspondre à une certain type de clusters, petits par exemple.

Par nature, le clustering ne permet pas de dissocier des alarmes en fonction des VMs ou des ressources à l'origine de l'alarme. La sémantique des clusters créés est donc à déterminer dans une seconde phase.

A partir de ce constat, nous avons été amenés à explorer une nouvelle méthode pour "caractériser" les clusters créés. Nous voulions notamment pouvoir répondre à la question de savoir si les clusters de comportement normal du système étaient "relativement proches" entre eux comparé à la localisation dans l'espace des clusters correspondant à des anomalies. Afin de répondre à cette question, l'étude du barycentre des clusters est alors parue pertinente.

3.3.4.2 Groupement des données

Pour cette technique de détection nous choisissons donc le groupement par-ressource (utilisée dans nos expérimentations préliminaires présentées dans les sous

sections précédentes). Nous avons considéré ce groupement afin de réduire le nombre de dimensions des observations à traiter par l'algorithme de clustering présent dans cette technique de détection. En effet, avec un nombre de dimensions important, il est peu aisé de déterminer si deux points sont relativement proches ou non [Hastie 2001]. De plus, réduire le nombre de dimensions de l'espace dans lequel chercher des clusters permet aussi de retrouver des clusters qui sont présents dans un espace de plus grande dimension. Cette assertion a été prouvée par Agrawal et al. dans leur travail présentant un nouvel algorithme de clustering dans des sous-espaces de données de grande dimension appelé CLIQUE [Agrawal 1998].

3.3.4.3 Phase de caractérisation de comportement

Les résultats préliminaires de notre étude de clustering nous ont amené à élaborer une *technique de caractérisation de comportement* d'un système par le biais d'un algorithme de clustering.

De la même manière que les observations d'un système traduisent son comportement et sont proches entre elles sans présence d'anomalie, cette technique repose sur les hypothèses suivantes :

Hypothèse 2 (H2): *Les clusters des observations d'un système traduisent son comportement.*

Hypothèse 3 (H3): *Les clusters d'observations de comportement normal d'un système sont proches entre eux.*

Hypothèse 4 (H4): *Les clusters d'observations d'anomalie sont éloignés des clusters de comportement normal.*

Cette technique se réalise par une première étape de clustering (comme décrit dans les expérimentations prélimianires) permettant la description de comportement proprement dite et une deuxième étape de calcul de descripteurs permettant la caractérisation des clusters identifiés à la première étape. La détection d'anomalies se réalise à partir de l'analyse de ces descripteurs.

Les deux étapes sont résumées dans la figure 3.3.3 pour l'étude d'une VM observée (la VM_1) du système cible représenté dans 3.2.3.

Elles supposent que toutes les observations d'un sous-flux qui arrivent à l'entité de détection sont affectées à des clusters, qu'elles correspondent à des instants d'anomalies ou non.

Étape 1 : tâche de clustering. L'étape de description de comportement traite les observations consécutives provenant d'un sous-flux par l'exécution d'un algorithme de clustering. Cet algorithme doit s'appliquer en ligne compte tenu des contraintes de détection dans les services cloud. Le choix de l'algorithme se tourne donc nécessairement vers la classe de clustering de flux de données (qui est la branche du clustering dédiée au traitement de données en ligne).

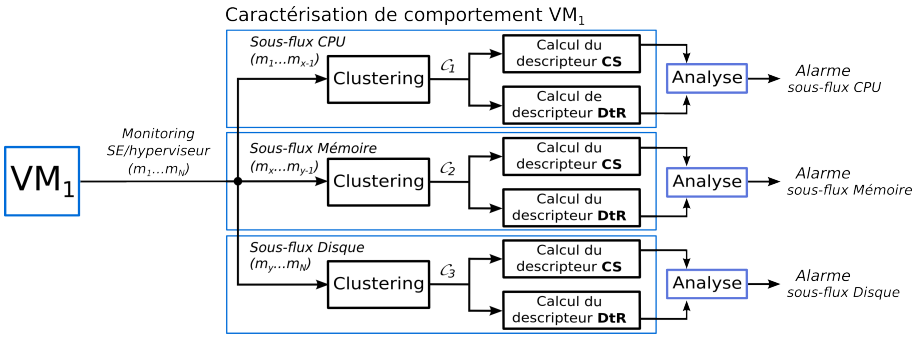


FIGURE 3.3.3 – Technique de caractérisation de comportement.

À chaque sous-flux est associé un ensemble de clusters décrivant les observations déjà traitées. Cet ensemble constitue le *modèle de clusters* du sous-flux. Sur réception d'une nouvelle observation d'un sous-flux, le modèle du sous-flux considéré est actualisé. Autrement dit, l'observation est affecté à une cluster, qu'elle corresponde à une comportement normal ou bien une anomalie. L'algorithme de clustering en ligne utilisé pour implémenter cette étape n'est pas imposé par notre technique (nous avions utilisé rEMM dans nos expérimentations prélimianires). Cet algorithme se doit toutefois de respecter les spécifications suivantes :

— Chaque observation reçue est soit affectée à un cluster existant soit affectée à un nouveau cluster dont elle est le centre.
— L'algorithme doit donner un poids aux clusters : les clusters assignés avec le plus d'observations doivent avoir le poids le plus important.
— Une fonction de vieillissement des clusters doit être appliquée, par exemple à un instant t le poids d'un cluster peut être vieilli de : $2^{-\lambda}$ avec $\lambda > 0$ une constante (comme dans rEMM [Hahsler 2010]). Ainsi les clusters qui n'ont pas d'observation affectée dans les itérations récentes de l'algorithme ont un poids réduit par rapport aux clusters nouvellement créés ou venant d'obtenir d'une observation.
— Enfin, les clusters de poids trop faible doivent pouvoir être supprimés du modèle. La suppression peut se faire sur la base d'un poids minimal toléré que nous appelons par la suite *tprune*. Ces clusters représentent en effet un état passé du système qui n'est plus d'actualité. Cela peut être dû à une reconfiguration du système ou bien à un changement significatif de type ou d'intensité de charge pour le système. Si le système revient à l'état caractérisé par des clusters effacés du modèle, des clusters similaires seront nouvellement créés en temps voulu.

Étape 2 : calcul de descripteurs. L'étape de calcul de descripteurs vise à caractériser l'évolution des clusters créés et mis à jour à l'étape précédente. Dans cette étape nous ne faisons pas de lien direct entre le nombre de clusters identifiés dans un sous-flux de données et le nombre d'états du système. Un état sémantique

du système (par exemple inactif, traitement d'une requête A ou en attente d'un réponse X) peut être représenté par plusieurs clusters. De même, une anomalie peut être représentée par 1 ou plusieurs clusters.

Cette étape repose sur le calcul du barycentre des clusters d'un sous-flux. Ce barycentre est un vecteur qui est la moyenne pondérée des centres de clusters. La pondération se fait avec le poids respectif des clusters.

À chaque instant correspondant à une nouvelle mise à jour du modèle de clusters, le barycentre est calculé selon l'équation suivante :

$$Bar(\mathcal{C}_{\mathcal{X}}) = \frac{1}{S}\sum_{i=1}^{n} w_i * v_i, \tag{3.1}$$

$$S = \sum_{i=1}^{n} w_i,$$

où n est le nombre de clusters dans $\mathcal{C}_{\mathcal{X}}$, v_i est le vecteur de coordonnées du centre du i^{ème} cluster (dépendant de l'implémentation de l'algorithme de clustering) dans l'espace de dimension du sous-flux et w_i le poids du i^{ème} cluster.

Nous considérons le barycentre comme la position représentant l'ensemble des clusters associés à un sous-flux à un temps donné. Tous les clusters (qu'ils contiennent des observations de comportement normal ou d'anomalie) sont considérés lors du calcul du barycentre. Il est à noter que des observations de comportement normal et d'anomalies peuvent avoir été affectées à un même cluster. Cette situation a deux causes probables.

— La première est que l'observation d'anomalie est proche dans l'espace des observations normales récentes. Dans ce cas, le barycentre et les descripteurs ne sont pas modifiés significativement lors d'une anomalie. L'anomalie ne sera peut-être pas détectée ou bien elle le sera si elle dure, mais avec un peu de latence.

— La seconde est qu'après une anomalie, les métriques du système mettent quelque temps à se stabiliser et certaines observations de comportement normal sont toujours assez proches des clusters d'anomalie.

Deux mesures sont calculées à partir de ce barycentre :

— La distance entre le barycentre et le cluster qui en est le plus proche, appelée $dist_min$.

— La distance entre le barycentre et le cluster qui en est le plus éloigné, appelée $dist_max$.

À partir du barycentre ainsi que des deux mesures, deux descripteurs sont finalement calculés. Conjointement avec le barycentre, ces descripteurs caractérisent le profil de dispersion des clusters dans l'espace de dimension du sous-flux.

— **Le descripteur DtR** (pour Distance to Reference en anglais). Il correspond à la distance entre le barycentre et un point de référence. Il représente la tendance des clusters à se mouvoir en groupe vers différentes régions de l'espace. Ce point de référence peut être le vecteur nul (souvent appelé origine de l'espace considéré). Il peut aussi être une observation mise à jour

périodiquement toutes les semaines, ou encore le vecteur correspondant à la moyenne sur une fenêtre de temps des observations collectées.

— **Le descripteur CS** (pour Cluster Spread en anglais). Il correspond à la différence entre *dist_max* et *dist_min*. Il représente la dispersion des clusters autour du barycentre (nous pouvons considérer cela comme la taille du voisinage des clusters d'un modèle).

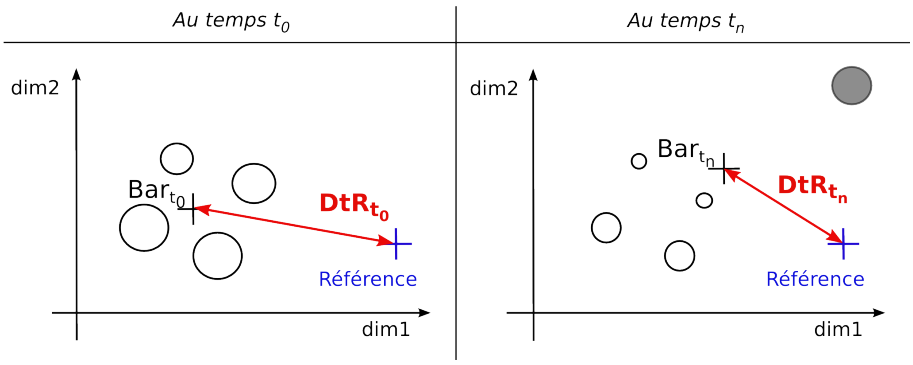


FIGURE 3.3.4 – Attribut DtR représenté pour un espace de 2 dimensions.

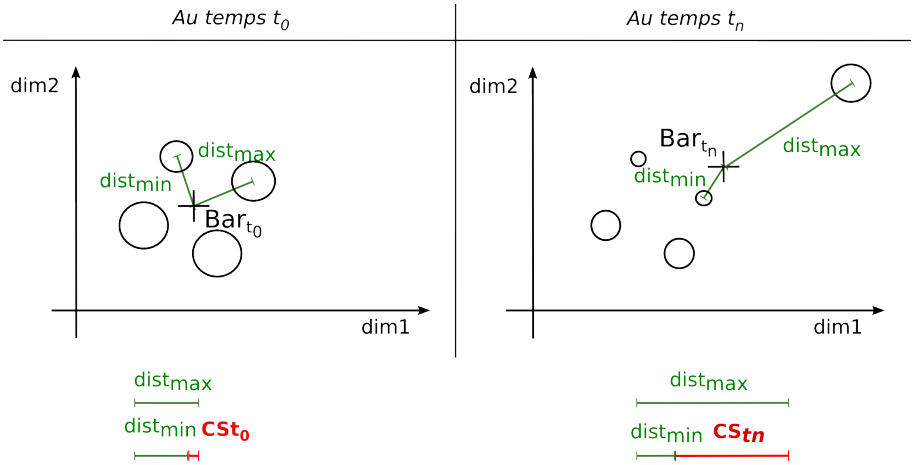


FIGURE 3.3.5 – Attribut CS représenté pour un espace de 2 dimensions.

En résumé, le barycentre ainsi que les descripteurs nous permettent d'étudier la mouvance globale de tous les clusters du modèle à un instant donné. Nous n'étudions donc pas le mouvement de chaque cluster individuellement.

Vue d'ensemble des deux étapes. Pour résumer ces deux étapes, sur chaque réception d'une observation d'un sous-flux $\mathcal{X}$, l'ensemble des clusters $\mathcal{C}_\mathcal{X}$ représentant le sous-flux sont dans un premier temps actualisés. Le barycentre des clusters est par la suite calculé. Enfin les descripteurs DtR et CS sont mis à jour en fonction des nouveaux poids des clusters.

3.3.4.4 Test et discussion à propos de la caractérisation de comportement

Grâce à notre expérimentation préliminaire, nous avons observé deux points. Premièrement, les deux descripteurs DtR et CS ont un faible écart type (noté et) au court du temps lors d'un comportement sans anomalie du système. Nous remarquons qu'au moins 95% des valeurs de CS calculées sont comprises dans l'intervalle $[-3et; 3et]$ où et est l'écart type de CS calculé sur le jeu de données de l'expérimentation préliminaire, pour les sous-flux disque, CPU et mémoire. Pour le sous-flux réseau cet intervalle contient 91% des valeurs.

Deuxièmement, nous avons constaté que l'injection d'erreurs dans notre système modifie effectivement la valeur des descripteurs. Ces injections seront détaillées dans le chapitre 4 décrivant la mise en œuvre de notre stratégie de détection. La figure 3.3.6 donne au lecteur un aperçu graphique de l'impact que peuvent avoir des injections sur la valeur du descripteur DtR calculé pour un sous-flux correspondant aux compteurs liés au disque d'une VM du système. Le point de référence de DtR est le vecteur nul de l'espace des compteurs liés au disque de la VM. Les injections ont lieu aux instants représentés par une valeur de DtR notée en couleur dans la figure (les points sont noir dans le cas contraire). Les couleurs différentes correspondent à des injections de type différent. Cette figure montre aussi que les différentes anomalies ne semblent pas affecter de la même manière le descripteur en ordre de grandeur.

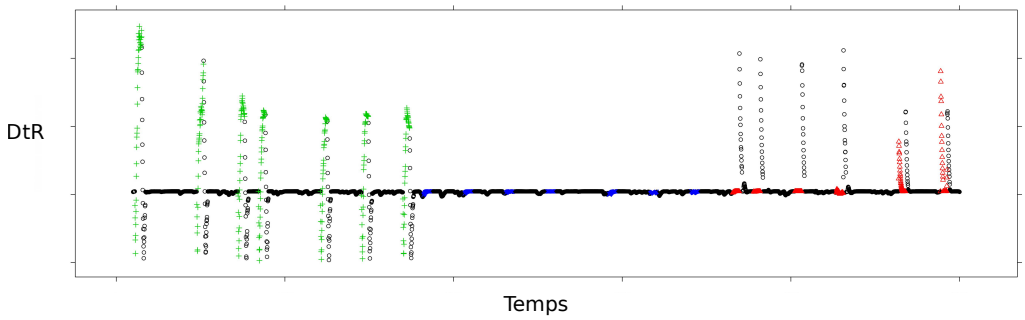


FIGURE 3.3.6 – Attribut DtR pour le sous-flux disque, en fonction du temps durant des tests d'injection d'anomalie.

L'analyse des variations des descripteurs dans chaque sous-flux permet donc de bien observer l'évolution des clusters et du comportement d'un système.

Nous faisons maintenant l'hypothèse suivante :

Hypothèse 5 (H5): *Une anomalie se traduit dans les valeurs de CS et de DtR par une variation significative.*

Discussion du descripteur DtR. D'après l'hypothèse H3, les créations, suppressions et déplacements de clusters de comportement normal ne provoquent pas une variation majeure dans la position du barycentre. Cela est valable aussi dans le cas où le voisinage des clusters de comportement normal se déplace lentement par rapport à un point de référence. Dans ce cas, le déplacement est dû à des évolutions lentes de charge de travail ou de consommation de ressources. Ce déplacement n'implique pas de variation majeure du descripteur DtR. Les variations majeures de DtR sont provoquées par la création de clusters éloignés du voisinage de clusters de comportement normal, qui sont donc des clusters d'anomalie (voir H4).

Discussion du descripteur CS. Un barycentre dont la position varie peu n'implique pas nécessairement que les clusters de comportement normal ne se dispersent pas autour du barycentre. Toutefois nous pouvons dire que des variations brutales dans les valeurs du descripteur CS correspondent à deux possibles évolutions du modèle : 1) la création de nouveaux clusters contenant des observations d'anomalies (ou clusters aberrants), 2) la suppression d'un cluster très proche du barycentre. Le premier cas présente que le descripteur CS est un indicateur d'anomalie par détection de création de clusters aberrants. Le deuxième cas de figure présente une faiblesse du descripteur CS en tant qu'indicateur d'anomalie car celui-ci peut donc varier sans création de clusters aberrants. C'est pour cette raison que l'étude de l'évolution du descripteur CS seul aboutit à de fausses alarmes qui peuvent être évitées en analysant en plus l'évolution du descripteur DtR.

3.3.4.5 Phase d'analyse de comportement

L'évolution dans le temps des deux descripteurs permet de représenter un sous-flux d'une VM par deux séries temporelles. Celles-ci sont caractérisées par leur sensibilité aux changements de comportement et leur stabilité lors de changements de charge de travail.

On distingue plusieurs méthodes pour l'analyse de ces descripteurs, par exemple :

— par analyse de valeur moyenne avec une fenêtre glissante,
— par des détections de rupture (en anglais Change Point detection),
— par des algorithmes de classification.

Cette analyse prend place au niveau de la boîte *Analyse* de la figure 3.3.3.

Dans le cas d'une analyse par algorithmes de classification supervisée, on parle de technique hybride. Elle combine en effet des algorithmes d'apprentissage supervisé et non supervisé. Il est en effet possible d'appliquer un algorithme à apprentissage supervisé sur la base des deux descripteurs CS et DtR. La capacité des descripteurs à s'adapter à l'évolution d'un système en restant constants sans présence d'anomalie, mais à varier lors d'une anomalie est exploitable par une classification performante afin de distinguer une anomalie dans la variation des descripteurs. Ainsi, un classifieur doit seulement apprendre à discriminer une déviation importante des deux

descripteurs afin de détecter une anomalie. Elle n'a donc pas à apprendre ce qui caractérise réellement une anomalie en termes de compteurs de performance.

Une technique hybride peut donc être réalisée à partir de chaque paire de descripteurs représentant un sous-flux. On parle alors de *détection hybride par sous-flux*. Elle peut également se réaliser sur l'ensemble des paires de tous les sous-flux d'une VM. Dans ce cas on parle de *détection hybride agrégée*.

3.3.4.6 Présentation de la technique en pseudo-code

L'algorithme 1 présente les deux étapes de la phase de caractérisation de comportement qui amènent à l'actualisation périodique des descripteurs CS et DtR, pour une VM observée et un sous-flux particulier.

À la suite d'une exécution de ces deux étapes pour un sous-flux donné $\mathcal{S}$, une alarme propre à $\mathcal{S}$ peut être levée grâce à l'analyse des descripteurs de $\mathcal{S}$. Celle-ci peut être levée à partir de l'analyse des descripteurs au cours du temps.

L'algorithme 2 présente la phase d'analyse implémentée par une méthode de levée d'alarme la plus simple. Cette méthode commence par analyser chacun des descripteurs de $\mathcal{S}$ séparément, grâce à une fenêtre glissante. Une déviation importante de la valeur d'un descripteur amène à une alarme locale pour ce descripteur. Une alarme pour $\mathcal{S}$ est levée lorsque l'analyse de l'un ou l'autre des descripteurs amène à une alarme locale au descripteur. Il y donc un OU logique fait entre les alarmes obtenues par les deux descripteurs.

Algorithme 1 Phase de caractérisation de comportement pour un sous-flux d'une VM du système cible

Input : VM : VM observée,
$metrics$: noms des compteurs du sous-flux,
$model$: ensemble de clusters représentant le sous-flux,
log_file : fichier de sortie actualisé périodiquement avec les valeurs CS et DtR courantes
while $True$ **do**
  if wait_monitoring_period() **then** ▷ L'attente est bloquante
    $obs \Leftarrow get_substream_metrics(VM, metrics)$
    $model \Leftarrow update_model(model, obs, \lambda, tprune)$
    $Bar \Leftarrow update_Bar(model)$
    $dist_{max} \Leftarrow compute_farther_cluster_distance(Bar)$
    $dist_{min} \Leftarrow compute_nearest_cluster_distance(Bar)$
    $DtR \Leftarrow compute_distance_Bar_to_refpoint(Bar)$
    $CS \Leftarrow compute_cluster_spread(model, Bar)$
    $write(CS, DtR, log_file)$
  end if
end while

Algorithme 2 Phase d'analyse des descripteurs

Input : DtR, CS : integer
  $alarme \Leftarrow 0$
  while $True$ **do**
    $DtR_alarm \Leftarrow get_rolling_window_alarm_DtR(DtR)$
    $CS_alarm \Leftarrow get_rolling_window_alarm_CS(CS)$
    $alarm \Leftarrow DtR_alarm \wedge CS_alarm$
    $write(alarm, log_file)$
  end while

3.3.4.7 Choix des paramètres de clustering

La vitesse de déplacement du barycentre dans l'espace est maîtrisée par *tprune* et λ. L'évolution globale des clusters est donc contrôlée par ces deux paramètres. Globalement, plus la valeur de ces paramètres est élevée plus vite le barycentre tend à se rapprocher de nouveaux clusters. En effet un fort vieillissement des clusters et un poids minimum élevé définissant le seuil d'élagage avancent la date de suppression des clusters. Ces clusters élagués n'étant plus pris en compte dans le calcul du barycentre, ce dernier finit rapidement par ne représenter que les clusters les plus récents.

En d'autres termes, λ et *tprune* contrôlent dans quelle mesure notre technique tolère les changements de comportement d'un système. Si l'on veut pouvoir tolérer de fortes variations de charge de travail (autrement dit ne pas considérer comme alarmant des changements abrupts de charge), un barycentre ne doit pas se rapprocher trop rapidement du nouveau voisinage de clusters correspondant à une charge de travail différente de la précédente. Il faut pouvoir atténuer les déplacements du barycentre dans les cas de changements de charge de travail sous réserve de toujours pouvoir détecter des anomalies. Il est intéressant de noter que deux méthodes significatives de paramétrage de λ et *tprune* sont facilement envisageables.

— La première consiste à opter pour une grande valeur pour λ et *tprune*. Elle aboutit à un modèle avec un nombre réduit de clusters car tout cluster ne contenant pas de nouvelles observations est rapidement supprimé du modèle. Dans ce cas, le barycentre se déplace rapidement vers de nouveaux clusters et s'habitue à de nouvelles charges de travail très rapidement. Toutefois, le nombre de clusters étant restreint, les nouvelles observations sont facilement à l'origine de nouveaux clusters. (Ces nouveaux clusters peuvent même dans certains cas correspondre à des clusters venant tout juste de se faire élaguer du modèle.) Par nature un nombre restreint de clusters définit un voisinage restreint. Ainsi de nouveaux clusters peuvent toujours paraître éloignés de ce voisinage restreint. Cette méthode aboutit donc aussi à de fausses alertes de détection de clusters aberrants.

— La seconde méthode consiste à sélectionner de très petites valeurs pour λ et *tprune*. Cette méthode fait en sorte que le modèle contienne un grand nombre de clusters représentant tout type de comportements passés du système.

Ainsi, à chaque fois que le système présente un comportement déjà observé, aucun nouveau cluster aberrant n'est créé. Lorsqu'une anomalie est détectée et est avérée, un mécanisme de suppression automatique des derniers clusters dans le voisinage aberrant identifié doit être activé. Toutefois, le nombre important de clusters dans le modèle ralentit la mise à jour de celui-ci pour chaque réception d'une nouvelle observation.

Nous donnons ici une illustration du choix de λ. Nous considérons que le système cible peut avoir un comportement qui évolue et dont les états représentatifs sont différents toutes les 10 min. Nous considérons de plus une collecte d'observations du système de 4 observations par minute. L'évolution des états représentatifs du système se fait progressivement, toutefois, il est possible d'imposer qu'un cluster (qui représente donc une partie d'un état) qui ne se voit pas affecté d'observation durant 10 min soit élagué du modèle de clusters. Il ne représente en effet plus l'état courant du système. Pour cela il suffit de configurer λ afin de diminuer le poids à t_x d'un tel cluster après 40 observations de manière à ce que le poids à t_{x+40} soit diminué d'un certain pourcentage.

La figure 3.3.7 représente le pourcentage de diminution du poids d'un cluster créé à un instant t_0 et ne recevant pas de nouvelle observation jusqu'à t_{40} en fonction de λ.

Dans une implémentation de cette technique, λ pourrait être variable et dépendre du nombre d'utilisateurs du service. Un nombre important d'utilisateurs amène le comportement du système à évoluer plus vite et à laisser des clusters obsolètes en moins de 10 min par exemple.

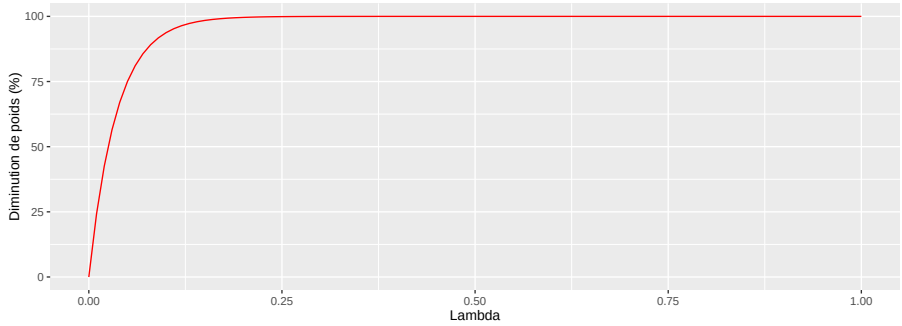


FIGURE 3.3.7 – Diminution du poids d'un cluster sans nouvelle observation en fonction de λ.

3.4 Conclusion

Dans ce chapitre nous avons présenté la cadre conceptuel de notre étude. Nous avons de plus défini les deux entités de notre stratégie de détection. Premièrement notre entité de monitoring a été définie avec ses deux sources de monitoring et ses deux méthodes de groupement de données. Deuxièmement notre entité de détection

a été définie avec nos niveaux de détection, nos types de détection ainsi que nos techniques de détection. Trois niveaux de détection sont proposés : 1) la détection d'erreurs, 2) la détection de symptômes préliminaires à une violation de service, et 3) la détection de violations de service. De plus la détection peut être réalisée de trois types différents : une détection binaire d'anomalie et deux détections multi-classes permettant le diagnostic du type d'erreur à l'origine de l'anomalie et le diagnostic de la VM qui est à l'origine de l'anomalie. Enfin les techniques définies sont basées respectivement sur de l'apprentissage supervisé et sur une caractérisation de comportement au moyen de clustering.

La mise en œuvre de cette stratégie sur une plateforme d'expérimentation est décrite dans le chapitre suivant.

Chapitre 4

Mise en œuvre

Sommaire

4.1 Plateforme d'expérimentation **60**
4.2 Module de monitoring **60**
4.2.1 Outils de monitoring 61
4.2.2 Compteurs de performance 61
4.2.3 Période de monitoring 62
4.3 Module de charge de travail **62**
4.4 Module d'injection de fautes **62**
4.4.1 Erreurs injectées 63
4.4.2 Campagne d'injection 66
4.4.3 Autres types de perturbations 68
4.5 Module de détection **70**
4.5.1 Mise en œuvre des différents types de détection 70
4.5.2 Détection par apprentissage supervisé 71
4.5.3 Détection DESC 72
4.6 Évaluation **74**
4.6.1 Exploitation des données 74
4.6.2 Évaluation des performances 75
4.6.3 Analyse de compteurs de performance 75
4.7 Conclusion **76**

Dans ce chapitre nous décrivons la mise en œuvre des entités de notre stratégie de détection présentées dans le chapitre précédent. Celle-ci nécessite le déploiement de plusieurs modules sur une plateforme d'expérimentation, à savoir un module de monitoring, un module de simulation d'une exécution réelle du système cible, un module d'injection de fautes utile à la collecte d'observations de comportement normal et d'erreur en un temps réduit, et un module de détection d'anomalies. Notre plateforme est constituée d'un cluster de deux serveurs. Les VMs nécessaires au déploiement des différents modules sont déployées sur ce cluster et représentées dans la figure 4.1.1. Nous décrivons de plus les méthodes utilisées pour l'évaluation des performances de cette stratégie.

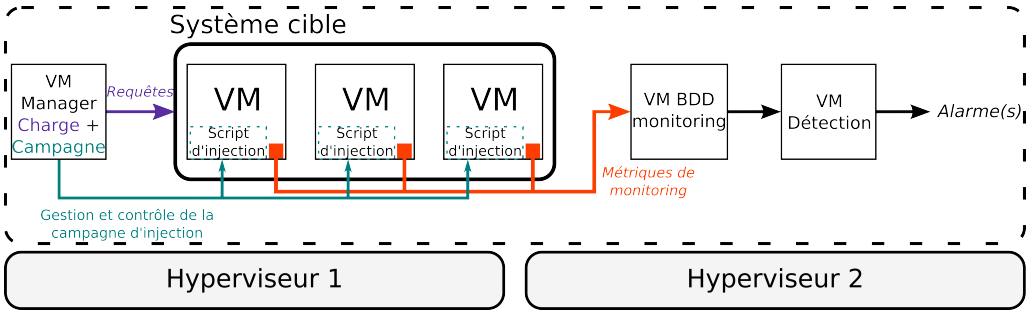


FIGURE 4.1.1 – Plateforme d'expérimentation.

4.1 Plateforme d'expérimentation

Afin de répondre à nos objectifs et d'évaluer les performances de détection de notre stratégie de détection d'anomalies, nous avons mis en place une plateforme de cloud privé. Ce cloud repose sur la solution VMware vCloud Suite [vCloud Suite 2015]. Au début des travaux de cette thèse, cette solution était considérée la plus complète et stable sur le marché et la plus souvent utilisée par les entreprises [1]. C'est pour cette raison que cette solution a été retenue pour le projet SVC dans lequel les travaux de cette thèse s'inscrivent [2]. La solution VMware vCloud Suite comprend VMware vCenter, vCloud Director et vShield Manager. La plateforme est composée de quatre machines physiques :

— Un serveur Dell PowerEdge R320 muni d'un CPU Intel Xeon E5-2407, de 8 Go de mémoire et de deux disques durs de 2 To. Il est utilisé en tant que serveur NFS pour le stockage des disques durs virtuels des VMs de ce cloud privé.

— Deux serveurs Dell PowerEdge R620 munis chacun de deux CPUs Intel Xeon E5-2660, de 64 Go de mémoire et de deux disques durs de 146 Go. Ils exécutent chacun un hyperviseur VMware ESXi 5.1.0 et hébergent les composants de la solution VMware vCloud Suite sous forme de VMs (ils sont appelés Hyperviseur 1 et 2 dans la figure 4.1.1).

— Un commutateur réseau HP 5120-24G EI. Il interconnecte les trois serveurs.

Chaque VM utilisée afin de déployer un composant d'un système cible possède 2 CPU, 10 Go de mémoire et 10 Go de disque. Elles sont connectées entre elles sur un réseau de 100 Mbps.

4.2 Module de monitoring

Le monitoring d'un système cible est orchestré dans nos travaux par deux sources de monitoring possibles, SE ou hyperviseur. Selon les deux sources de monitoring, les outils de collecte utilisés et compteurs de performances collectés sont différents.

1. https ://virtualizationreview.com/blogs/the-schwartz-cloud-report/2014/06/cloud-survey.aspx (vu octobre 2016)
2. Projet français d'Investissements d'Avenir Secured Virtual Cloud (SVC) (2012-2015)

Ils sont présentés dans la suite avec la période de monitoring choisie pour les deux sources.

4.2.1 Outils de monitoring

Dans le cas du monitoring SE, les données de monitoring sont obtenues grâce au système de monitoring pour systèmes distribués Ganglia [Massie 2003]. Ganglia est populaire dans les communautés de calcul à haute performance (en anglais High Performance Computing), big data et cloud [3] car il est non intrusif et peut gérer des clusters de machines jusqu'à 2000 nœuds. Il repose sur une structure hiérarchique en arbre. Des démons installés dans les nœuds collectent des données qu'il est possible de personnaliser grâce à des modules écrits en python. Ils correspondent aux feuilles de l'arbre. Des "méta" démons, collectent les données des démons de monitoring installés sur les nœuds d'un même cluster. Les méta démons peuvent collecter des données d'autres méta démons afin de centraliser les données d'une infrastructure de plusieurs clusters par exemple. Dans nos travaux, nous installons un agent Ganglia dans chaque VM d'un système cible. Les données de monitoring sont centralisées par un méta démon installé dans une VM externe au système cible.

Pour le monitoring hyperviseur, les données de monitoring sont collectées par le biais de l'API des hyperviseurs VMware. Un script de collecte des métriques de monitoring est exécuté périodiquement. Il utilise la librairie python Pysphere [4].

Dans les deux cas, les données de monitoring sont envoyées en base de données par le biais d'un réseau de management dédié. La base de données est hébergée par la *VM BDD monitoring* représentée figure 4.1.1. Lors des expérimentations, les données collectées sont stockées dans une base de données PostgreSQL [5].

4.2.2 Compteurs de performance

Grâce aux deux sources SE et hyperviseur, il nous est possible de collecter de nombreux compteurs de performance. Les listes des compteurs de performances sont données en annexe A. Ils sont 216 pour le monitoring SE et 152 pour le monitoring hyperviseur. Il existe des compteurs qui restent invariants sans modification venant d'un opérateur sur le système cible. C'est le cas de certains compteurs liés à des ressources matérielles. Des exemples sont la vitesse CPU, le nombre de CPUs, la taille d'un disque ou bien la taille d'une mémoire. Ces compteurs ne sont pas utilisés pour définir les dimensions d'observations sur lesquelles s'appliquent les algorithmes d'apprentissage automatique que nous traitons dans nos travaux. Ils sont toutefois utilisés dans le but de ramener des compteurs en unités de pourcentages. Par exemple la mémoire disponible d'un système sera exprimée en pourcentage de la taille totale de cette mémoire.

3. http ://www.admin-magazine.com/HPC/Articles/Monitoring-HPC-Systems (vu octobre 2016)
4. https ://pypi.python.org/pypi/pysphere (vu octobre 2016)
5. http ://www.postgresqlfr.org/ (vu octobre 2016)

4.2.3 Période de monitoring

Les données de monitoring sont collectées avec une période de 15 secondes. Pour des raisons de taille de stockage ainsi que de temps de traitement des données nous n'avons pas appliqué une période plus courte. Ce point n'impacte toutefois pas les conclusions tirées de nos travaux. Nous injectons en effet directement des erreurs sur une durée supérieure à la période de monitoring. Lors de l'application de nos méthodes dans un cas réel, une périodicité de 1 seconde est toutefois recommandée afin que la détection se fasse le plus proche possible de l'instant de départ de l'erreur.

4.3 Module de charge de travail

Tout cas d'étude confondu, une charge de travail est nécessaire à la simulation d'un environnement opérationnel sur notre plateforme. Cette charge simule un environnement réel d'exploitation du service cloud considéré. Selon les cas d'étude, la charge de travail est composée de requêtes pour des protocoles différents. Elle est donc mise en œuvre par un outil dont l'implémentation dépend du type de service proposé par le système cible. Cet outil est déployé sur une VM de notre plateforme appelée *VM Manager* dans la figure 4.1.1.

Une charge de travail est employée dans deux buts différents lors d'une expérimentation.

— Dans un premier temps, elle est nécessaire à la phase d'entrainement des modèles de détection à apprentissage supervisé dans le but de pouvoir collecter des données d'entrainement pour les modèles.
— Dans un deuxième temps, elle est nécessaire pour nos expérimentations afin d'évaluer les performances de notre stratégie de détection lors de la phase de détection des techniques utilisées.

Dans notre étude elle est de plus utile car elle nous indique le nombre de requêtes ayant été traîtées ou non par le système cible à chaque instant. Nous en déduisons le PE (ou pourcentge d'échec) pour chaque instant et donc chaque observation de monitoring collectées. Ce point est utile dans le but de positionner des labels nécessaires au fonctionnement des algorithmes à apprentissage supervisé sur nos observation. Cette tâche est présentées dans la sous section 4.5.1.

4.4 Module d'injection de fautes

Nous rappelons que le déploiement d'un module d'injection de fautes possède une double fonction.

— La première fonction est de construire des jeux de données d'entrainement pour des algorithmes à apprentissage supervisé avec des entrées représentant à la fois des comportements normaux et anormaux. C'est seulement avec un jeu de données comportant des entrées avec et sans anomalie que les modèles peuvent apprendre à classifier les entrées de nouveaux jeux de données lors d'une phase de détection. Cette technique nous permet de réaliser la collecte

de jeu de données pour apprentissage supervisé de manière méthodique et dans un temps d'observation restreint. Nous savons ainsi précisément dans quels cas d'anomalies nos évaluations sont généralisables. De plus, plusieurs mois, voire des années d'observation du système seraient nécessaires avant que des anomalies ne s'activent.

— La deuxième fonction concerne l'évaluation de notre stratégie de détection. L'injection de fautes nous permet d'émuler la présence d'anomalies en phase de détection sur notre plateforme expérimentale et dans un temps d'observation restreint.

Dans nos travaux, nous nous focalisons plus particulièrement sur l'injection d'erreurs menant à des anomalies des différents niveaux de détection présentés (voir 3.3.1). Cela correspond à une vue plus haut niveau du système que l'injection de fautes [6]. Ces erreurs sont injectées de manière logicielle pour des questions de flexibilité d'implémentation de la méthode d'injection.

Alors que les fautes logicielles et matérielles sont très nombreuses et que les défaillances sont le plus souvent liées à des combinaisons de plusieurs fautes [Yuan 2014], il peut être très complexe d'établir une couverture complète des erreurs traitées. Toutefois, il se trouve que de nombreuses erreurs dues à des combinaisons de fautes sont connues. Ces erreurs se traduisent par une structure particulière du système qu'il est possible de caractériser par l'analyse des compteurs de performance système.

Dans cette section nous détaillons les types d'erreurs que nous injectons dans un système cible ainsi que leur implémentation.

4.4.1 Erreurs injectées

Nous considérons dans notre modèle d'erreur l'injection de pics d'utilisation de ressources système. Les fautes émulées par ces pics représentent des fautes présentes aussi bien dans la partie logicielle que matérielle d'un système. Elles peuvent correspondre à des fautes de développement ou de configuration temporaires internes ou externes du système [Avizienis 2004].

Les erreurs injectées peuvent potentiellement mener à des défaillances intermittentes. Elles sont de cinq types différents : 1) CPU, 2) mémoire, 3) disque, 4) perte de paquets, 5) latence réseau (également appelé latence).

4.4.1.1 Causes émulées et moyens utilisés

Les **injections CPU** sont mises en œuvre en lançant un programme sollicitant un certain pourcentage de la CPU libre de la VM cible et particulièrement son Unité Arithmétique et Logique (UAL). Cette activité est dupliquée selon le nombre de cœurs de la machine considérée. Elle est de durée déterminée et est configurée

6. Dans un but de diagnostic de fautes, il serait nécessaire de travailler au plus proche de la cause réelle en injectant des fautes et non pas des erreurs.

pour occuper un pourcentage prédéfini de l'activité de chaque cœur durant cette durée.

L'activité injectée émule par exemple un programme potentiellement bloqué dans une boucle algorithmique dont la condition de sortie est mal définie. Certaines instructions contenues dans la boucle sont alors réalisées un nombre potentiellement infini de fois. L'existence d'un timer ou de toute autre règle d'exécution que le programme se doit de vérifier peut forcer celui-ci à sortir de cette boucle. Le timer peut être mal programmé (i.e. trop long ou reposant sur le test d'une condition erronée) ou inexistant.

Les **injections mémoire** sont mises en œuvre en allouant un certain pourcentage de la mémoire libre de la VM cible. Des blocs de mémoires sont réservés puis accédés aléatoirement durant le temps d'injection.

Ces injections reproduisent les conséquences sur un système d'un programme possédant des fuites mémoire. La mémoire allouée dans un tel programme peut ne pas être systématiquement libérée. Un programme peut également s'avérer bloqué dans une boucle allouant indéfiniment de la mémoire.

Les **injections disque** sont mises en œuvre par deux actions. Elles provoquent la synchronisation (en anglais flush) des données en mémoire tampon sur le périphérique disque. Leur deuxième tâche consiste à créer des descripteurs de fichiers ainsi que des vecteurs de 50 Mo en mémoire. Les descripteurs de fichiers sont affectés à des vecteurs d'octets et une opération de synchronisation de ces descripteurs est réalisée afin d'écrire chaque vecteur stocké en mémoire tampon sur le périphérique disque.

L'activité injectée émule des programmes ou des pilotes coincés dans des tâches ou boucles de lecture et d'écriture et de synchronisation sur disque. Un nombre de lectures trop important peut être dû à des scans intempestifs d'anti-virus par exemple. Une telle activité peut également émuler un système de fichiers corrompu et incluant des boucles dans la structure de certains répertoires. Un autre exemple est celui de l'activité de mécanismes enclenchés par la détection de défaillances disque, visant à allouer de nouveau des données à des secteurs disque non corrompus.

Les erreurs disques provoquent des ralentissements au niveau du système et une faible activité du CPU car les écritures sur des périphériques disque sont des tâches généralement longues.

Les injections de perte de paquets et de latence réseau sont mises en œuvre en manipulant des paquets IP (Internet Protocol).

— Les **injections de perte de paquets** imposées sur les données entrantes (en ingress) et sortantes (en egress) émulent des défaillances sur le réseau de communication de la VM cible. Elles occasionnent des problèmes qui se répercutent sur les VMs destinataires des paquets supprimés en sortie de la VM cible ainsi que les VMs émettrices de paquets supprimés en entrée de la VM cible. Elles peuvent être dues à des débordements de mémoire tampon de routeurs, de switchs, de VM émettrices ou de la VM cible, par manque de mémoire ou bien car les périphériques réseau n'émettent/transmettent pas les paquets assez rapidement.

— Les **injections de latence réseau** ajoutent des délais d'envoi et de réception de paquets. Ces délais émulent des capacités de calcul de VMs ou de routage de routeurs insuffisantes dues à un matériel trop vieux ou non mis à l'échelle. Ils peuvent également être dus à des canaux réseau vieillissants ou à pertes de paquets obligeant des réordonnancement de paquets coûteux en temps.

Les injections sont orchestrées grâce à des outils de manipulation de paquets réseau pour noyaux Linux. Le premier outil est *iptables*. Il est l'interface utilisateur de configuration du framework de définition de pare-feux *Netfilter*. Le second est *tc* du package *iproute2* qui permet de gérer les structures de contrôle du trafic réseau des noyaux Linux (i.e. les queuing discipline (qdisc), les classes et les filtres). Ces deux outils gouvernent des appels à des fonctions du noyau Linux dont l'action est localisée sous la couche réseau du modèle OSI.

4.4.1.2 Réflexion sur le diagnostic

Les différentes types d'erreur peuvent notamment être détectées par les catégories de compteurs qui décrivent une ressource système sollicitées par la mise en œuvre de l'injection. Les erreurs émulées par des injections de CPU peuvent être détectées par les compteurs liés à la CPU ; les erreurs émulées par injections mémoire peuvent être détectées par les compteurs liés à la mémoire ; les erreurs émulées par injections disque peuvent être détectées par les compteurs liés au disque ; enfin, les erreurs émulées par injections de perte de paquets et de latence réseau peuvent potentiellement être détectées par les compteurs liés au protocole TCP (Transmission Control Protocol) ou UDP (User Datagram Protocol) ainsi que les compteurs liés au protocole IP.

Les compteurs TCP permettent en particulier de bien détecter les pertes de paquets. En effet TCP est un protocole qui intègre des mécanismes permettant la vérification et la gestion de nouveaux envois de paquets perdus. Ce type d'évènement est géré par le mécanisme d'évitement de congestion de TCP. Du point de vue d'une instance protocolaire TCP émettrice, une perte de paquets est traduite par une réduction de bande passante (dont les causes peuvent être multiple telle que des tampons mémoire plein et un lien réseau venant d'être déprécié). Dans un tel cas, l'instance de protocole émetteur réduit la taille de la fenêtre d'émission de paquet afin de palier ce problème. Ainsi une injection de perte de paquets peut impacter par exemple le nombre de segments TCP traités par seconde par la couche transport de la machine destination (i.e. celui-ci se retrouve diminué), le nombre de segments retransmis par la machine source (i.e. celui-ci se retrouve augmenté), ou le nombre de segments envoyés par la machine source (i.e. celui-ci se retrouve augmenté).

4.4.2 Campagne d'injection

4.4.2.1 Description

Une campagne d'injection correspond à l'exécution d'un script principal paramétrable qui effectue périodiquement des injections dans un système cible. Pour cela, ce script principal invoque des scripts d'injection installés dans les VMs du système cible afin d'effectuer les injections qu'il est paramétré pour exécuter. Le script principal est déployé dans une VM de notre plateforme expérimentale qui est représentée par la *VM Manager* dans la figure 4.1.1.

Lorsqu'une campagne d'injection est exécutée, le monitoring (SE ou hyperviseur) est lancé afin de collecter des données du système cible. L'activation du monitoring et le lancement d'une campagne d'injection correspondent à une *expérimentation.*

Les paramètres d'une campagne sont les suivants et sont expliqués dans la suite :

— les VMs cibles des injections, *l_vm*,
— les erreurs à injecter, *l_erreur*,
— l'intensité des erreurs à injecter, *l_intensite*,
— la durée d'injection, *duree_injection*,
— le temps de pause séparant chaque occurrence d'injection, *pause*.

Lors d'une campagne, chaque erreur de chaque intensité mentionnée en paramètre est injectée dans chaque VM également passée en paramètre. Chaque erreur de chaque intensité est d'abord injectée dans une première VM, puis dans une deuxième, etc. L'algorithme 3 définit une campagne d'injection en pseudo-code. De plus, le schéma en figure 4.4.1 illustre les temps d'injection pour une campagne avec $l_erreur = \{CPU\}$, $l_intensity = \{4, 7\}$ et $l_vm = \{VM_1, VM_2\}$. Une campagne s'arrête lorsque toutes les injections de chaque intensité ont été injectées dans les VMs. Les pauses entre les injections permettent au système cible de se stabiliser dans un comportement normal après une injection. Le monitoring durant une campagne d'injection collecte ainsi des observations de comportement normal ainsi que des observations d'anomalie.

Algorithme 3 Campagne d'injection

Input : VM : *l_vm*, *l_erreur*, *l_intensite*, *duree_injection*, *pause*

```
for vm in l_vm do
    for err in l_erreur do
        for intens in l_intensite do
            injection = Injection(err, intens, duree_injection)
            inject_in_vm(vm, injection)
            sleep(pause)
        end for
    end for
end for
```

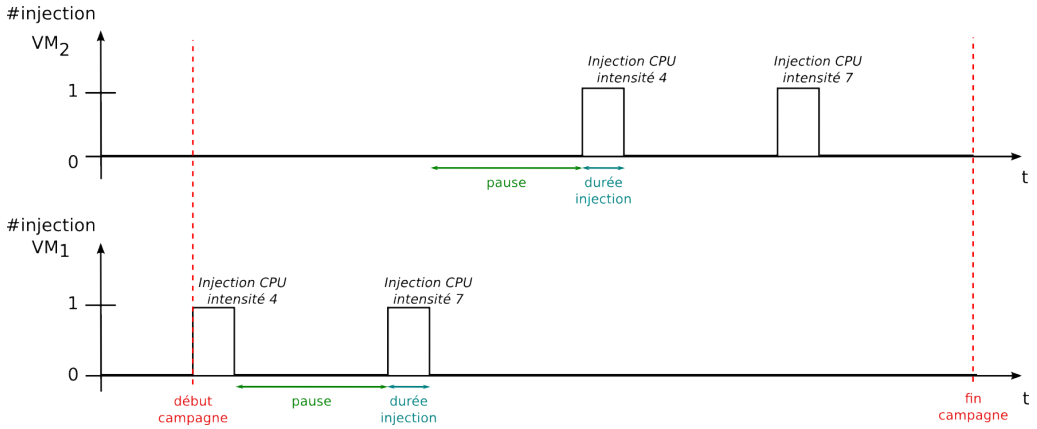


FIGURE 4.4.1 – Diagramme illustrant une campagne d'injection.

4.4.2.2 Intensité d'injection

Les injections sont calibrées par un niveau d'*intensité* qui correspond à une gradation de l'ampleur d'une erreur. Selon le type d'erreur, une intensité donnée peut donner lieu à des défaillances plus ou moins graves. Dans nos travaux, les injections de faible intensité émulent des défaillances bénignes. Les injections de forte intensité n'émulent pas nécessairement des défaillances catastrophiques car un service cloud peut ne pas être sensible à une type d'erreur en particulier, quelque soit son intensité d'injection.

Les différentes intensités d'injection sont présentées dans le tableau 4.1. À titre explicatif, nous détaillons chaque type d'erreur pour une intensité donnée. L'injection CPU d'intensité 1 est mise en œuvre en monopolisant 30% de la CPU inutilisée d'une VM cible. L'injection mémoire d'intensité 1 est mise en œuvre en monopolisant 79% de la mémoire libre d'une VM cible. L'injection disque d'intensité 1 est mise en œuvre en créant 20 processus exécutant des synchronisations et des écritures disque sur une une VM cible. L'injection de perte de paquets réseau d'intensité 1 est mise en œuvre en supprimant 3.2% des paquets manipulés sur une VM cible. L'injection de latence réseau d'intensité 1 est mise en œuvre en ajoutant 32 ms de délai d'envoi et de réception pour un paquet IP.

Classe d'erreur	**Unité**	**Intensité**						
		1	**2**	**3**	**4**	**5**	**6**	**7**
CPU	%	30	40	50	60	70	80	90
Mémoire	%	79	82	85	88	91	94	97
Disque	#processus	20	25	30	35	40	45	50
Perte de paquets réseau	%	3.2	4.0	4.8	5.6	6.4	7.2	8.0
Latence réseau	ms.	32	40	48	56	64	72	80

Tableau 4.1 – Intensité des injections.

Les intensités d'injection ont été calibrées sur la base d'expérimentations préliminaires basées sur les cas d'étude que nous avons explorés. La calibration des intensités pour notre cas d'étude MongoDB est notamment présentée dans une de nos précédentes publications [Silvestre 2014].

Dans le cas des injections CPU, mémoire et disque, les intensités sont définies selon les ressources du SE des VMs. Autrement dit, l'intensité la plus haute (resp. basse) est la consommation maximum (resp. minimum) de ressource autorisée par le SE (resp. impactant le SE de manière à ce que des violations de service soient constatées sur au moins une des VMs). En effet, lors d'une injection à forte intensité et proche du maximum de ressource utilisable, le SE peut être amené à terminer les sous processus qui composent l'injection. Dans le cas des erreurs réseau, l'intensité maximale est calibrée de manière à amener le service à un pourcentage d'échac (PE) d'environ 99% pour son injection dans au moins une VM.

Ces expérimentations de calibrage nous permettent de conclure que les différentes injections n'ont pas le même effet en terme de nombre et de durée de violations de service selon la VM dans laquelle elles sont exécutées. Le calibrage est donc à adapter au cas d'étude et n'est pas nécessairement utilisable tel qu'il est sur une autre plateforme avec d'autres cas d'étude.

4.4.2.3 Durée d'injection

Le temps d'injection est choisi assez long afin de collecter un nombre suffisant d'observations d'erreurs ou de violations de service tout en restant assez court afin de pouvoir généraliser nos résultats à de courtes anomalies. De plus, des expérimentations préliminaires montrent que des injections pendant des périodes plus courtes ne mènent pas nécessairement à des violations de service ce qui nous empêche de collecter des observations intéressantes pour nos tests de détection. Une campagne d'injection avec des durées d'injection assez longues est donc nécessaire afin de collecter assez d'observations de violation de service dans le but de les étudier.

4.4.2.4 Temps de pause

Le temps de pause doit être choisi expérimentalement. Il permet au système cible de se stabiliser dans un comportement normal après une injection. Il permet également la collecte d'un nombre suffisant d'observations de comportement normal durant une expérimentation.

4.4.3 Autres types de perturbations

D'autres initiatives ont été menées dans le cadre de nos travaux concernant l'injection de fautes. Nous n'avons pas retenu ces initiatives et nous sommes concentrés sur les quatre classes d'injection présentées dans 4.4.1.

Nous avons travaillé tout d'abord à la mise en place de redirections de fonctions ou bien d'appels système (i.e. hook de fonction) au sein des services cloud. Dans ce cas, lors d'un appel à une fonction, la redirection vers un code espion peut

mener à l'exécution ou non de la fonction initiale ou bien à l'exécution de celle-ci en mode dégradé : permutation de paramètres, bit-flip dans un paramètre, renvoi de mauvais code d'erreur, exécution avec instructions manquantes (i.e., suppression d'instructions *free()* par exemple), etc.

Nous avons développé un injecteur de fautes permettant de réaliser de telles redirections lors de l'exécution d'un programme Open source grâce à la variable d'environnement *LD_PRELOAD* sur système Unix comme dans les travaux de Wierman et al. sur le framework Vajra [Wierman 2008]. Cette variable permet d'imposer à l'éditeur de liens le chargement d'une librairie pour l'exécution d'un programme avant toutes les autres librairies partagées. La librairie chargée peut par exemple sauvegarder le code de la fonction *free()* de la librairie *stdlib* et choisir de rediriger son adresse sur un code de fonctionnement personnalisé. De cette manière un code d'exécution dégradé de la fonction *free()* peut être exécuté sous certaines conditions alors que son code original peut être exécuté sous d'autres selon les instructions du code personnel substitué.

Nous avons de plus expérimenté l'injection de bit-flips dans la mémoire d'un hyperviseur, voulant ainsi émuler l'effet de fautes physiques liées à des rayonnements. Ces injections en mémoire ont été réalisées avec l'outil d'injection IronHide [Lone Sang F. 2012] développé sur une carte FPGA (pour Field-Programmable Gate Arrays) et communiquant sur le bus PCIE (pour Peripheral Component Interconnect Express). Dans le but de tests préliminaires nous avons étudié l'hyperviseur faisant tourner des VMs hébergeant une base de données et répondant constamment aux requêtes de la charge de travail d'un benchmark. Dans un premier temps, des bit-flips ont été injectés aléatoirement dans la mémoire de l'hyperviseur. À la suite d'une observation durant 48 h en suivant des injections, aucune dégradation de service n'a pu être observée à partir de l'analyse humaine des données de monitoring des VMs et du cluster d'hyperviseurs. Dans un second temps nous avons tenté de lire aléatoirement certaines plages d'adresses tout en les classifiant selon le type d'octets qu'elles pouvaient contenir. Cette classification de plage fut réalisée visuellement. Autrement dit, nous avons associé un pourcentage de gris à chaque valeur d'octet et la couleur rouge à chaque octet correspondant à un caractère ASCII. Nous avons ensuite représenté visuellement la mémoire avec des pixels colorés selon la valeur de chaque octet, localisant ainsi les zones mémoire vides, ou contenant du texte par exemple. Les bit-flips injectés aléatoirement dans ces zones n'a pas permis de conclure à un quelconque effet des bit-flips dans le système. Nous avons ensuite réduit l'injection de fautes à certains types de plages de stockage. Toutefois, aucun résultat probant n'a ici aussi été remarqué selon le même protocole d'injection individuelle et d'observation des services de l'hyperviseur sous 48 h. Enfin, nous avons aussi injecté des effacements d'octets dans la mémoire de celui-ci, voulant simuler l'écriture d'une donnée à une mauvaise adresse. Toutefois, dans ces deux derniers cas, aucune erreur ni défaillance n'a pu être remarquée selon le même protocole d'injections individuelles et d'observation des services de l'hyperviseur sous 48 h.

4.5 Module de détection

Nous présentons dans cette section la mise en œuvre de nos types et techniques de détection d'anomalie. Cette mise en œuvre constitue un module déployé dans une VM de notre plateforme expérimentale qui est représentée par la *VM Détection* dans la figure 4.1.1. Elle est dépendante des algorithmes d'apprentissage automatique que nous avons sélectionnés et que nous présentons dans cette section.

4.5.1 Mise en œuvre des différents types de détection

Lors des expérimentations, les données collectées par le monitoring du système cible sont datées et labelisées. Autrement dit, à chaque observation correspond une date ainsi que plusieurs labels positionnés automatiquement. Une observation possède autant de labels que de niveaux de service étudiés (voir 3.3.1), à savoir trois. La valeur des labels d'une observation indique si l'observation correspond à une VM qui contient une erreur d'un type donné, des symptômes préliminaires à une violation de service et une violation de service à la date de l'observation. Ces labels sont indispensables à la phase d'entrainement d'algorithmes à apprentissage supervisé ainsi que lors des tests d'évaluation de notre stratégie de détection afin de calculer les performances de détection (i.e. évaluer si lors d'une levée d'alarme, une anomalie a bien lieu ou non).

Cette labelisation est possible car nous avons la connaissance de quels instants correspondent à des instants lors desquels une injection à lieu grâce à l'implémentation de la campagne d'injection. De plus, nous connaissons les instants durant lesquels le PE (ou pourcentage d'échec) est supérieur à la limite PE_max grâce à la charge de travail. Autrement dit nous connaissons les instants durant lesquels le système amène une violation de service. Enfin, les labels de symptômes préliminaires à une violation de service sont positionnés automatiquement a posteriori de la collecte de données, après analyse des instants de violation de service, afin de labeliser les instants de la période $[t - \delta_t, t]$ où t est l'instant d'occurrence de la violation de service apparaît.

Lors de tests d'évaluation, un traitement des observations (en fonction de leurs labels et de la VM qu'elles décrivent) est nécessaire en amont de chaque test afin d'actualiser les labels selon le type de détection évalué par le tests (les types de détection sont présentés dans 3.3.2).

Il existe donc trois type de labelisation correspondant aux trois types de détection : labelisation binaire, labelisation multi-classe selon le type d'erreur à l'origine d'une anomalie, et labelisation multi-classe selon la VM source d'une anomalie.

— La *labelisation binaire* consiste à affecter le label 1 à chaque observation d'une VM dont la date correspond à une période d'injection d'erreur dans cette VM (respectivement, une période de violation de service ou période de symptômes préliminaires à une violation) et affecter le label 0 sinon.

— Pour la *labelisation multi-classe selon le type d'erreur*, un label est associé à chaque type d'erreur à partir de 1. Cette labelisation consiste à affecter

le label correspondant à une erreur à chaque observation d'une VM dont la date correspond à une période d'injection de l'erreur considérée dans cette VM (respectivement, une période de violation de service due à l'injection de l'erreur considérée). Le label 0 est affecté aux observations de comportement normal.

— Pour la *labelisation multi-classe selon la VM source de l'erreur*, un label est associé à chaque VM du système cible à partir de 1. Cette labelisation consiste à affecter le label correspondant à une VM donnée à chaque observation d'une VM dont la date correspond à une période d'injection d'une erreur, peu importe laquelle, dans la VM associée au label (respectivement, une période de violation de service due à une injection dans la VM considérée). Le label 0 est affecté aux observations de comportement normal.

Les labels sont utilisés durant les deux phases opérationnelles d'une classification à savoir, une phase d'entrainement et une phase de détection. Durant la phase d'entrainement ils sont utiles aux modèles à entrainer afin d'identifier les classes à discerner. Durant la phase de détection ils sont utiles à nos scripts d'évaluation afin de compter les observations d'un jeu de données dont le label prédit est bien le label réel.

4.5.2 Détection par apprentissage supervisé

Afin de mettre en œuvre la technique de détection par apprentissage supervisé, nous avons utilisé l'algorithme Random Forest [Breiman 2001]. C'est un algorithme connu pour son efficacité de détection ainsi que sa faculté à traiter des jeux de données avec un grand nombre d'attributs. L'algorithme Random Forest est basé sur plusieurs arbres de décision, chacun entrainé à prédire des classes de données sur un sous-ensemble du jeu de données d'entrainement. Il utilise la moyenne des prédictions des différents arbres afin d'augmenter ses performances par rapport à un simple arbre de décision et évite aussi le "surapprentissage" (lorsqu'un modèle est trop ajusté à un jeu de données et n'est pas efficace pour d'autres jeux de données). Chaque arbre de décision est entrainé avec un sous-ensemble des entrées du jeu de données d'entrainement (en anglais *bootstrapping*) ainsi qu'un sous ensemble des attributs de ce jeu de données (la taille de ce sous-ensemble est à donner en paramètre de l'algorithme). L'entrainement des arbres sur un sous-ensemble d'attributs réduit la corrélation de leurs prédictions.

Nous utilisons de plus d'autres algorithmes en comparaison au Random Forest dans l'évaluation que nous faison de cette technique dans le chapitre 5. Chaque algorithme est testé avec l'implémentation de la librairie python *scikit-learn*[7]. Lors de l'analyse des résultats, dans le cas où aucune mention n'est faite d'un paramètre d'un algorithme, la valeur par défaut proposée par la libraire est choisie.

Une opération de centrage/réduction est appliquée aux données d'entrainement. Cette opération consiste à soustraire sa moyenne à une métrique et la diviser par son écart type (la moyenne et l'écart type sont calculés à partir de toutes les observations

7. http ://scikit-learn.org/ (vu octobre 2016)

du jeu de données d'entrainement) et ce, pour toutes les observations du jeu de données d'entrainement. Les données de test sont par la suite centrées/réduites grâce aux mêmes valeurs de moyenne et d'écart type ainsi obtenues. Étant donné qu'elle conserve les tendances de variation des métriques, cette opération permet de comparer facilement des données sans problème d'échelle.

Une opération de centrage/réduction doit être appliquée dans le cas où les métriques n'ont pas les mêmes unités. En effet, en règle générale lors d'un traitement de données, toutes les métriques doivent être exprimées en unités fondamentales (sans unité dérivée). Cette opération n'est pas à effectuer lors de la comparaison de métriques de même unité fondamentale.

Prenons en exemple un jeu de 4 observations de deux métriques inspiré d'une présentation de l'école polytechnique fédérale de Zurich [8], représenté dans le tableau 4.2 (a). Les points A, B, C et D ont deux composantes exprimées dans les mêmes unités (en %). Les points A et C sont a priori proches car ils correspondent à un profil de consommation caractérisé par une consommation mémoire très élevée et une faible consommation CPU. De même pour les points B et D avec une consommation mémoire moyenne et une faible consommation CPU. Toutefois, une fois ces données centrées/réduites, il est impossible de discerner ces différents profils de consommation car les variations de l'occupation mémoire et de l'activité CPU qui permettaient de différencier les deux comportements sont maintenant similaires. Le tableau 4.2 (b) représente le même jeu de données mais cette fois centré/réduit et démontre bien ce point.

(a)

Point	Mem%	CPU%
A	100	7
B	30	7
C	100	10
D	30	10

(b)

Point	Mem	CPU
A	0.8660254	-0.8660254
B	-0.8660254	-0.8660254
C	0.8660254	0.8660254
D	-0.8660254	0.8660254

Tableau 4.2 – Exemple d'un jeu d'observations dans le premier cas exprimé en pourcentages et dans le deuxième cas avec les métriques centrées réduites.

Le déploiement de cette détection est donc court et avec un temps de configuration négligeable.

4.5.3 Détection DESC

L'implémentation de notre caractérisation de comportement utilisant le clustering est basée pour sa première phase d'exécution sur l'implémentation en R de

8. https ://stat.ethz.ch/education/semesters/ss2012/ams/slides/v4.2.pdf (vu octobre 2016)

l'algorithme de clustering rEMM [9] présenté dans les résultats préliminaires motivant le développement de DESC (voir 3.3.4). L'algorithme est configuré avec les paramètres par défaut, $\lambda = 0.1$ et $tprune = 0.4$. Plusieurs distances ont été testées dans une étude préliminaire. La distance euclidienne a été retenue pour nos évaluations car elle est rapide à calculer et donnait les meilleurs résultats. La deuxième phase de DESC (voir la sous section 3.3.4.5) a été testée à l'aide d'une implémentation simple de fenêtre glissante, ainsi qu'avec une analyse par apprentissage supervisé implémentée avec l'algorithme Random Forest présenté plus haut dans cette section. Le deuxième cas est appelé *DESC hybride*. Le cas d'une analyse par fenêtre glissante a été publié dans un de nos précédents travaux [Sauvanaud 2015]. Les résultats sont excellents en termes de précision, recal et taux de faux positifs pour une analyse sur un jeu de données avec une distribution de cas positifs et négatifs équilibrée.

C'est la détection DESC hybride que nous évaluons dans ce manuscrit.

Une opération de centrage/réduction est appliquée à certains attributs en fonction de la moyenne et de l'écart type des premières observations traitées par l'algorithme (nous avons choisi 100 correspondant à 25 min de monitoring de période 15 sec). En conditions opérationnelles, la moyenne et l'écart type devraient être actualisés tous les jours.

Cette opération permet de ne pas biaiser des calculs de distance. En effet, les métriques aux valeurs les plus larges ont le poids le plus important dans le calcul de distance. Les résultats d'un algorithme tel que le Nearest Neighbors peuvent donc être totalement bouleversés par un changement d'échelle des métriques ou un mauvais traitement des unités.

Quatre sous-flux de données sont sélectionnés afin d'y appliquer l'algorithme de clustering à savoir liés au CPU, à la mémoire, au disque et au réseau. Ces sous-flux ne sont définis arbitrairement. Les compteurs de performance qui les composent ne couvrent pas la majorité des compteurs disponibles pour la détection. Une étude plus poussée serait de définir davantage de sous-flux pour améliorer la détection.

Les métriques CPU sont étudiées sans centrage/réduction car chacune d'entre elles est exprimée en pourcentage. C'est aussi le cas pour les métriques de disque. Les métriques réseau et liées à la mémoire sont quant à elles centrées/réduites. Les compteurs de performance associés sont présentés en annexe A.1.2.

Le déploiement de la détection DESC reste donc court avec un temps de configuration négligeable, tout comme pour la détection par apprentissage supervisé.

Remarque : Une expérimentation avec l'algorithme non supervisé One Classe SVM [Schölkopf 2001] implémenté par la librairie scikit-learn a été réalisée. Aucun résultat acceptable n'a pu être obtenu sur nos données malgré une étude de sensibilité faite sur les paramètres.

9. https ://cran.r-project.org/web/packages/rEMM/index.html (vu octobre 2016)

4.6 Évaluation

4.6.1 Exploitation des données

Lors de l'exécution d'une expérimentation, nous collectons un jeu de données. Une fois que les données d'un jeu sont labelisées, il est possible de les utiliser pour exécuter des tests d'évaluation de performance de notre stratégie de détection d'anomalies.

Lors d'un test de performance de la détection par apprentissage supervisé, un ou deux jeux de données peuvent être utilisés.

— Dans le cas de l'utilisation d'un unique jeu de données, celui-ci est divisé en deux parties afin de constituer un jeu de données d'entrainement et un jeu de données pour l'évaluation des performances (i.e., pour la prédiction) : 40% des observations du jeu de données sont sélectionnés aléatoirement et utilisés comme jeu d'entrainement. Les 60% restant sont utilisés pour la prédiction.
— Dans le cas de l'utilisation de deux jeux de données différents, un jeu de données est alors utilisé pour l'entrainement de modèles et l'autre pour la prédiction.

L'évaluation de la détection DESC avec une phase d'analyse par analyse de valeur moyenne avec une fenêtre glissante ou par détection de rupture par exemple, ne peuvent se faire qu'avec un unique jeu de données. La suite temporelle des observations traitées est nécessaire à ces algorithmes. L'évaluation de la détection DESC hybride peut être réalisée de manière similaire à l'évaluation de la détection par apprentissage supervisé.

— Dans le cas où un seul jeu de données est utilisé, un modèle de clusters est actualisé et les descripteurs sont calculés pour chaque observations, prise de la plus ancienne à la plus récente, du jeu de données. Les valeurs des deux descripteurs qui sont calculées pour chaque observation (i.e. deux valeurs pour chaque observation) forment un unique nouveau jeu de données à analyser. Nous rappelons que la phase d'analyse est faite par un algorithme à apprentissage supervisé. Avant de procéder à cette phase, le nouveau jeu de données est divisé en deux parties, l'une pour l'entrainement et l'autre pour la prédiction de l'algorithme de cette phase.
— Dans le cas où deux jeux de données sont utilisés, un jeu de données est utilisé pour l'entrainement de modèles et l'autre pour la prédiction. Les descripteurs sont calculés pour chaque jeu de données. Deux nouveaux jeux de données sont ainsi obtenus. Ils sont par la suite utilisés respectivement pour l'apprentissage et pour la prédiction.

Sans mention particulière, pour chaque test les jeux de données collectés sont sous-échantillonnés aléatoirement de manière à ce que chaque classe définie pour la détection représente 2% du nombre d'observations de comportement normal. Ce pourcentage est choisi afin de considérer des résultats prenant en compte le fait que des anomalies sont des évènements rares.

Nous ajoutons de plus qu'une étude a été faite concernant l'application de l'al-

gorithme PCA sur les données. L'algorithme a été utilisé sur nos jeux de données afin d'identifier des composantes principales. La détection par apprentissage supervisé a été appliquée à partir du traitement de certaines des composantes principales identifiées. Les résultats de performance constaté étaient inférieur à ceux obtenu lors de tests sans appliquer l'algorithme PCA. Nous avons donc pu conclure qu'appliquer de cette manière cet algorithme amène à perdre de l'information utile à la détection. Sachant de plus que l'application de PCA sur les données amènent une couche d'abstraction supplémentaire à la compréhension de la création des modèles de détection, nous n'avons pas retenu cette méthode pour poursuivre nos travaux.

4.6.2 Évaluation des performances

L'analyse des résultats d'évaluation de performances est présentée au moyen des mesures ROC AUC et PR AUC présentées dans le chapitre 2. Cette présentation des résultats sous forme de plusieurs mesures de performances complémentaires donne une représentation plus complète des performances de détection. Dans notre contexte, nous considérons que l'étude des courbes ROC est nécessaire mais pas suffisante pour conclure des performances de détection. Nous souhaitons en effet maximiser le taux de vrais positifs et minimiser le taux de faux positifs tout en observant si la précision et le rappel associé restent élevés. Pour cela, les courbes PR sont donc également nécessaires à l'étude des performances.

Les performances sont dites *excellentes* lorsque les mesures d'AUC (ROC ou PR) moyennes sont supérieures à 0.90, *acceptables* lorsque qu'elles sont comprises entre 0.90 et 0.70, et *inutilisables* lorsqu'elles sont inférieures à 0.70.

Les résultats de tests sont présentés sous forme de diagrammes en boîte à moustaches. Pour obtenir ces diagrammes, 100 modèles créés sont testés, chacun reposant sur une sélection aléatoire d'attributs pour créer des arbres de décision (voir la description de l'algorithme Random Forest dans 4.5.2), et sur une sélection aléatoire des observations du jeu d'entrainement du modèle considéré. Ces multiples exécutions permettent une comparaison plus juste des résultats de performance. Nous n'avons donc pas réalisé de validation croisée afin de faire apparaître plus clairement sur les courbes les variations de performances.

4.6.3 Analyse de compteurs de performance

La description des compteurs collectés via l'API VMware est donnée dans la documentation en ligne de notre solution cloud VMware [10].

Le nom des compteurs est formé de la manière suivante $famille_nom$ ou $famille$ est la famille du compteur considéré et *nom* est le nom du compteur considéré. Il existe plusieurs familles de compteurs que nous regroupons dans 5 groupes que sont les compteurs mémoire, réseau, disque CPU et énergie (nous utilisons les groupes prédéfinis des documentations VMware et Ganglia). Nous notons pour la

10. https ://www.vmware.com/support/developer/converter-sdk/conv61_apireference/vim.PerformanceManager.html (vu octobre 2016)

compréhension des tableaux que le groupe CPU concentre les familles *cpu*, *load*, et *rescpu* où les compteurs *rescpu* et *load* sont exprimés comme des moyennes de temps d'activité lorsque $famille = rescpu$. Le groupe disque concentre les familles *disk*, *diskstat* et *datastore* où les compteurs *datastore* représentent les accès disque en général sur une centralisation de ressource disque (i.e., sans le détail des accès disque par disque). Le groupe mémoire concentre les familles *mem*, *swap* (utilisation de l'espace de swap), *proc* (gestion des processus) et *Overhead* (gestion de la mémoire d'une VM par l'hyperviseur). Enfin, le groupe réseau concentre les familles *net*, *tx* (transmission de paquets ou d'octets), *rx* (réception de paquets ou d'octets), *tcp* et *tcpext* (lié au protocole TCP et collecté par la commande *netstat*).

Les tableaux ont le code couleur suivant pour les différents groupes : (compteurs mémoire : rouge), (compteurs réseau : vert), (compteurs disque : magenta), (compteurs CPU : bleu) et (compteurs énergie : orange).

4.7 Conclusion

Dans ce chapitre nous avons décrit les modules nécessaires à notre stratégie de détection d'anomalies ainsi que leur mise en œuvre sur une plateforme d'expérimentation. Nous avons de plus décrit les méthodes et mesures que nous utilisons pour l'évaluation des performances de détection de cette stratégie. Cette évaluation repose sur l'utilisation de jeux de données de monitoring collectées à partir du système cible de détection et lors de campagnes d'injection.

Une partie de cette mise en œuvre a été publiée sous la forme d'un outil de détection dans [Silvestre 2015b].

Nous déployons deux cas d'études sur notre plateforme : un système de gestion de base de données ainsi qu'une fonction réseau virtualisée. Le premier cas d'étude ainsi les tests d'évaluation de notre stratégie menées sur celui-ci sont présentés dans le chapitre suivant.

Chapitre 5

Évaluation du cas d'étude SGBD : MongoDB

Sommaire

5.1 Présentation **78**
5.1.1 Description et déploiement 78
5.1.2 Charge de travail 79
5.1.3 Jeux de données 79
5.2 Évaluation de la détection par apprentissage supervisé **80**
5.2.1 Comparaison d'algorithmes : performances de détection . . 80
5.2.2 Comparaison d'algorithmes : temps d'exécution 81
5.2.3 Comparaison d'algorithmes : durée de validité du jeu d'entrainement 82
5.2.4 Détection d'erreurs 82
5.3 Évaluation de la détection DESC hybride **83**
5.3.1 Détection d'erreurs 83
5.3.2 Durée de validité du jeu d'entrainement 85
5.4 Analyse des compteurs de performance utiles à la détection **88**
5.5 Conclusion **89**

Ce chapitre présente l'évaluation expérimentale de notre stratégie de détection pour le cas d'étude d'un système de gestion de base de données (SGBD) très populaire MongoDB. Il est spécialement développé pour le dimensionnement dynamique et la rapidité d'accès aux données ce qui en fait un cas d'étude pertinent pour un service cloud.

Ce cas d'étude est simple et constitue le point de départ de notre recherche. Les tests menés pour ce cas servent comme base pour une évaluation plus approfondie que nous faisons pour un deuxième cas d'étude dans le chapitre suivant.

Nous mettons ici en évidence des tests à mettre en œuvre qui sont nécessaires à la collecte méthodique d'un jeu de données d'entrainement complet et dans un

temps restreint. Nous mettons de plus en évidence des tests pour une validation rigoureuse d'une stratégie de détection d'anomalie. Nous effectuons également des tests afin de vérifier si un diagnostic de l'origine de l'anomalie est possible en même temps que la détection de l'anomalie. Chaque test mené nous permet également de conclure sur les performances de détection de notre stratégie pour une détection par apprentissages supervisé et une détection DESC hybride.

L'analyse des compteurs de performance utiles à la détection d'anomalies lors de ces tests est également faite. Elle nous permet de comprendre de quelle manière les erreurs sont détectées et donc de quelle manière elles se manifestent dans notre système cible. Nous analysons ainsi si un "dictionnaire d'anomalies" peut être mis en place. Autrement dit, nous cherchons à déterminer si nous pouvons généraliser l'usage de certains compteurs de performance dans la détection d'un certain type d'erreur ou dans la détection dans une VM en particulier.

5.1 Présentation

5.1.1 Description et déploiement

MongoDB est un SGBD open source NoSQL (qui ne repose pas sur le modèle relationnel) destiné à la gestion de documents. Une collection de documents peut être partitionnée et distribuée sur plusieurs serveurs afin de conserver un débit d'accès important aux documents tout en ne saturant pas un seul serveur. Une partition de données peut elle-même être répliquée sur plusieurs serveurs afin de garantir la disponibilité des données. Un routeur redirige les requêtes faites à la base de données selon la partition concernées par ces requêtes.

Le nœud *primaire* d'une partition de données reçoit toutes les requêtes en écriture et par défaut reçoit aussi toutes les requêtes en lecture destinées à cette partition. Les nœuds *secondaires* répliquent régulièrement toutes les opérations exécutées par le nœud primaire de la partition à laquelle ils appartiennent et écrites en log. Un nœud secondaire peut être élu nœud primaire par le routeur lorsque le nœud primaire ne répond plus aux requêtes assez rapidement.

Notre déploiement comprend une base de documents séparées en deux partitions. Chaque partition comprend un nœud primaire et deux nœuds secondaires. Les nœuds primaires et secondaires de notre déploiement sont la cible d'injections d'erreurs.

Il est représenté Figure 5.1.1 et comprend six VMs. Les VMs sont numérotées de 1 à 6 dans la suite. Les VMs affectées à une partition restent affectées à cette partition sans configuration manuelle imposant le contraire. Toutefois, si la VM1 appartient à la partition 1, et qu'elle correspond à la VM primaire de cette partition à l'instant t, elle peut correspondre à une VM secondaire de cette même partition à l'instant $t + n$ dans le cas où elle ne répond plus assez rapidement aux requêtes.

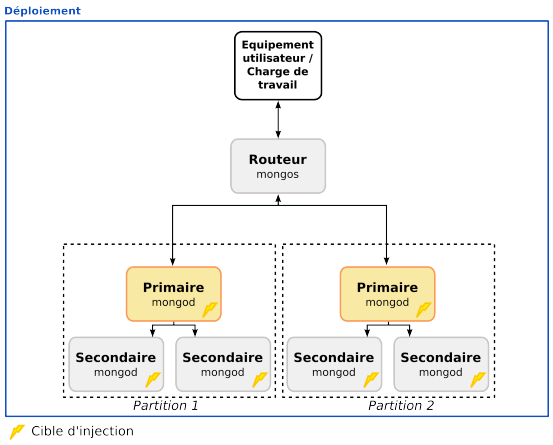


FIGURE 5.1.1 – Déploiement MongoDB.

5.1.2 Charge de travail

Nous utilisons la charge de travail Yahoo! Cloud Serving Benchmark [Cooper 2010] afin de simuler les conditions opérationnelles de MongoDB. Pour nos expérimentations, nous exploitons deux types de charge :

1. Charge fixe : Envoi de 3000 requêtes READ par seconde à la base de données.
2. Charge variable : Envoi d'un nombre entre 500 et 3000 requêtes READ et UPDATE par seconde à la base de données, ce nombre étant sélectionné aléatoirement toutes les 10 min.

5.1.3 Jeux de données

5.1.3.1 Description

Nous avons effectué deux expérimentations qui nous ont permis de collecter deux jeux de données d'environ 90000 observations chacun (i.e. environ 15 jours de monitoring) appelés $\mathcal{Y}$ et $\mathcal{Z}$. Chacun a été collecté lors de l'exécution d'une campagne d'injection à partir de notre plateforme MongoDB.

Pour chaque campagne, nous avons les paramètres communs suivants :

— $l_vm = \{VM1, VM2, VM3, VM4, VM5, VM6\}$,
— $l_erreur = \{CPU, mémoire, disque, latence, perte_paquet\}$,
— $l_intensité = \{1 : 7\}$ (soit 210 injections),
— $d_injection = 10$ min,
— $pause = 100$ min,
— La collecte des données est faite par monitoring hyperviseur.

$\mathcal{Y}$ est collectée grâce à une expérimentation avec une charge fixe. $\mathcal{Z}$ est collectée grâce à une expérimentation avec une charge variable et qui a été lancée 5 mois après l'expérimentation de $\mathcal{Y}$. Entre la collecte de $\mathcal{Y}$ et de $\mathcal{Z}$, nous avons utilisé notre plateforme pour d'autres tests qui ne sont pas exploités dans cette thèse. Aucune mise à jour logicielle n'a été effectuée.

Nos choix de configuration sont motivés par les points suivants :

— Les données ont été collectées en monitoring SE afin d'obtenir un ensemble de données avec le plus grand nombre de compteurs de performance.

— Nos précédents travaux montrent l'étude du cas MongoDB avec des injections de 3 jours [Silvestre 2014]. Nous voulons évaluer ici si nos résultats sont généralisables à des durées d'injection plus courtes.

5.1.3.2 Utilisation

Dans la présentation des résultats qui suit, lorsqu'il n'est pas fait mention du jeu de données utilisé lors d'un test, le jeu de données $\mathcal{Y}$ est utilisé.

Bien que les données de chaque VM de notre déploiement aient été exploitées, les résultats de performance de détection sont présentés pour une de nos deux VMs primaires. Les résultats sont similaires pour les autres VMs du cas d'étude et ne sont pas présentés.

5.2 Évaluation de la détection par apprentissage supervisé

Dans cette section nous présentons les résultats obtenus pour la détection d'erreurs en utilisant une détection par apprentissage supervisé. Un travail précédent a été mené pour ce cas d'étude et concernait la détection de violations de service selon le type d'erreur locale à la VM observée [Silvestre 2015b].

La détection est testée avec plusieurs algorithmes très populaires afin de comparer leurs performances de détection d'anomalies. L'algorithme Random Forest est l'algorithme qui mène aux meilleures performances de détection et c'est cet algorithme qui est utilisé par la suite pour les tests réalisés par détection DESC hybride.

L'étude des performances de détection pour une prédiction qui a lieu sur des données collectées 5 mois après la collecte des données d'entrainement est également faite. Elle nous permet d'évaluer au bout de combien de temps un modèle cesse de donner des résultats acceptables.

La détection par apprentissage supervisé implémentée avec Random Forest est abrégée *Detect_Sup_RF*. La détection DESC hybride est abrégée *Detect_DESC_hybride*.

5.2.1 Comparaison d'algorithmes : performances de détection

Une classification binaire d'erreur a été réalisée afin de comparer différents algorithmes à apprentissage supervisé. Les figures 5.2.1 (a) et (b) présentent les ROC et PR AUC calculées pour la détection binaire d'erreur.

Quatre algorithmes ont été testés :

— Random Forest (configuration 200 arbres),

— Réseau de neurones (configuration avec une fonction de tangente hyperbolique (comme dans [LeCun 1991]), une fonction softmax, et un taux d'apprentissage de 0.001),
— Nearest Neighbors (configuration avec $k = 3$),
— Classification naïve Bayésienne (en anglais Naive Bayes) (configuration par défaut de la librairie sklearn référencée dans la sous section 4.5.2).

Les algorithmes SVM et Gradient Boosting ont été évalués dans un de nos précédents travaux [Silvestre 2015b], et ne sont pas présentés.

(a) ROC AUC. (b) PR AUC.

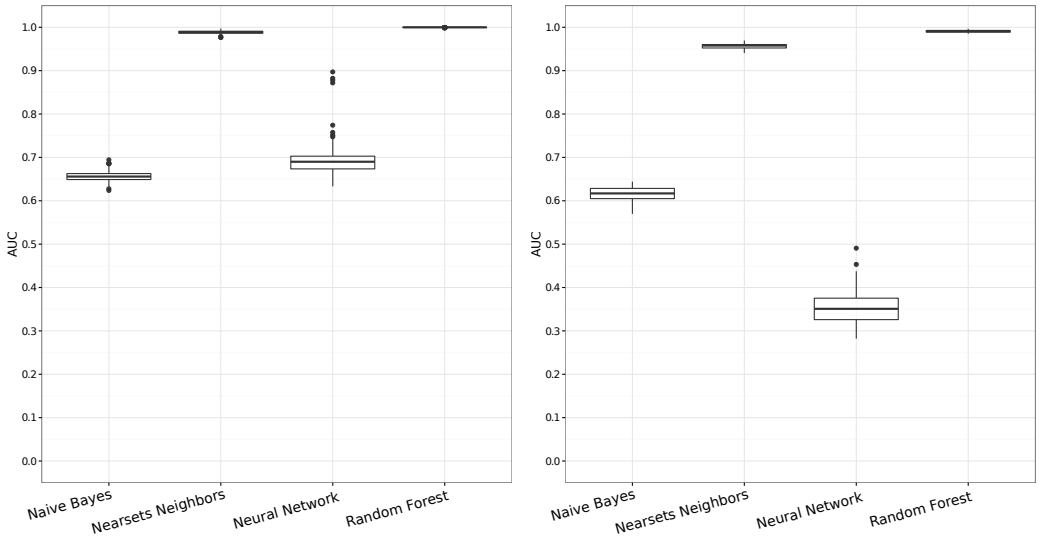


FIGURE 5.2.1 – Détection d'erreurs binaire avec plusieurs algorithmes à apprentissage supervisé.

D'après les résultats obtenus, les meilleurs algorithmes pour la détection s'avèrent être Random Forest et Nearest Neighbors. Random Forest possède toutefois une PR AUC moyenne d'environ 0.99, qui est supérieure à celle du Nearest Neighbors qui est d'environ 0.95.

5.2.2 Comparaison d'algorithmes : temps d'exécution

Le tableau 5.1 présente les temps moyens de traitement pour une observation (de 216 compteurs) par ces mêmes algorithmes en phase d'entrainement et en phase de détection, exprimés en milliseconde. Les temps sont calculés sur une machine avec un processeur 64bits Intel Core i7 2.10GHz. Les moyennes sont calculées sur 100 exécutions de détection.

L'algorithme Neural Network possède un temps d'entrainement dix fois supérieur en moyenne par rapport aux autres algorithmes. Son temps de prédiction est toutefois faible avec 0.168 ms en moyenne. L'algorithme Naive Bayes est l'algo-

rithme le plus intéressant à utiliser pour son temps d'entrainement de seulement 0.010 ms et de prédiction de 0.15 ms. L'algorithme Nearest Neighbors a lui le temps de prédiction le plus long avec 0.82, ce qui reste un temps acceptable dans notre contexte. Enfin, l'algorithme Random Forest exhibe un temps d'entrainement de 0.919 ms et un temps de prédiction 10 fois inférieur. Ces temps sont plus importants que les temps des algorithmes Naive Bayes et Nearest Neighbors mais tout en restant bas et acceptables pour notre contexte.

En considérant les deux algorithmes qui ont fourni les meilleures performances de détection, nous concluons que Random Forest, avec des temps d'entrainement et de prédiction faibles, est l'algorithme le plus intéressant à utiliser en ligne.

		Naive Bayes	**Neural Network**	**Random Forest**	**Nearest Neighbors**
Entrainement	Moyenne	0.010	10.322	0.919	0.064
	Écart type	0.004	1.824	0.130	0.017
Prédiction	Moyenne	0.015	0.168	0.102	0.82
	Écart type	0.005	0.045	0.021	0.193

Tableau 5.1 – Temps d'entrainement et de prédiction (ms).

5.2.3 Comparaison d'algorithmes : durée de validité du jeu d'entrainement

Les figures 5.2.2 (a) et (b) présentent les ROC et PR AUC calculées pour la détection binaire d'erreurs avec un entrainement sur $\mathcal{Y}$ et une prédiction sur $\mathcal{Z}$. L'algorithme Random Forest exhibe de meilleures performances, que ce soit du point de vue de la mesure des ROC AUC ou des PR AUC. En effet la ROC AUC moyenne pour Random Forest est supérieure à 0.90 alors qu'elle ne dépasse pas 0.65 pour le reste des algorithmes.

De plus la PR AUC moyenne est de 0.66 alors qu'elle est entre 0.25 et 0.51 pour le reste des algorithmes.

Beaucoup d'erreurs ne sont néanmoins pas détectées avec une prédiction qui a lieu sur des données collectées 1 mois après la collecte des données d'entrainement. La qualité des prédictions se dégrade donc si l'on ne fait pas un entrainement régulièrement. L'avantage de ce cas de figure est qu'avec l'algorithme Random Forest, si peu d'erreurs sont détectées, les administrateurs ne sont pas de surcroît contraints à traiter de fausses alarmes.

5.2.4 Détection d'erreurs

Les figures 5.2.3 (a) et (b) ainsi que les figures 5.2.4 (a) et (b) représentent les ROC et PR AUC pour la détection d'erreurs selon le type d'erreur respectivement pour une étude de $\mathcal{Y}$ et de $\mathcal{Z}$. Nous étudions ici seulement l'algorithme Random Forest.

(a) ROC AUC. (b) PR AUC.

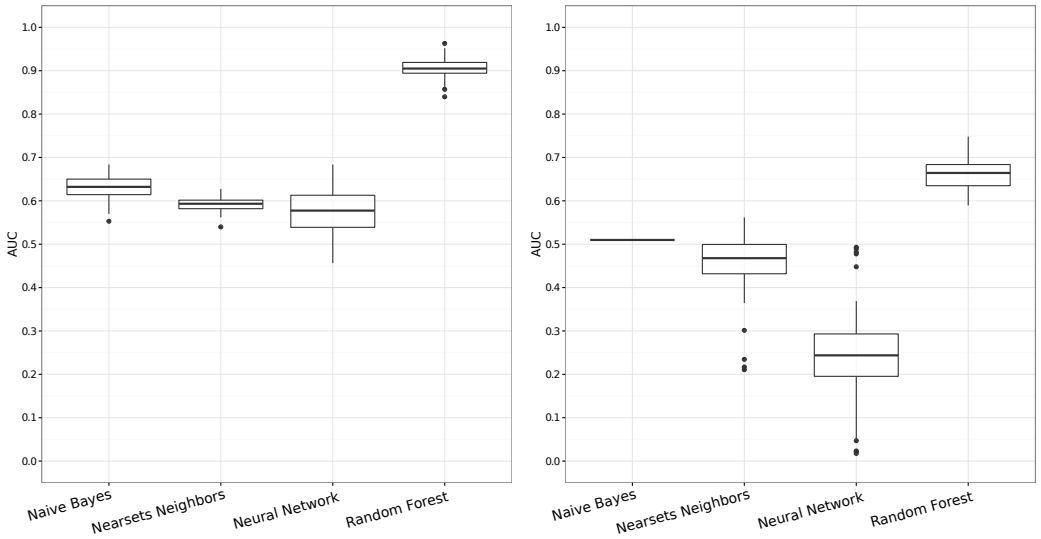


FIGURE 5.2.2 – Détection d'erreurs binaire avec plusieurs algorithmes à apprentissage supervisé et pour un entrainement sur $\mathcal{Y}$ et une prédiction sur $\mathcal{Z}$.

Les performances sont excellentes pour l'analyse des deux jeux de données avec des ROC et PR AUC moyennes supérieures à 0.99. La charge variable exécutée lors de la collecte de $\mathcal{Z}$ n'affecte quasiment pas les performances de détection.

5.3 Évaluation de la détection DESC hybride

Nous évaluons ici les performances de détection DESC hybride appliquée avec comme algorithme de classification Random Forest. L'analyse des données se fait pour une détection hybride par sous-flux et agrégée (voir l'explication en 3.3.4).

La détection d'anomalies menée est de type multi-classe selon le type d'erreur locale à la VM observée. L'étude des performances de détection pour une prédiction qui a lieu sur des données collectées 5 mois après la collecte des données d'entrainement est également faite.

5.3.1 Détection d'erreurs

Les figures 5.3.1 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs avec DESC hybride par sous-flux, selon le type d'erreur locale à la VM observée. Les figures montrent toutes que les erreurs sont détectées avec des résultats acceptables par au moins un sous-flux en considérant les ROC et de PR AUC. Les erreurs CPU, disque et mémoire sont très bien détectées par le sous-flux dont les compteurs sont liés à leur type respectif. L'erreur de latence réseau est mieux détectée par le sous-flux mémoire avec une ROC AUC moyenne de 0.90 et

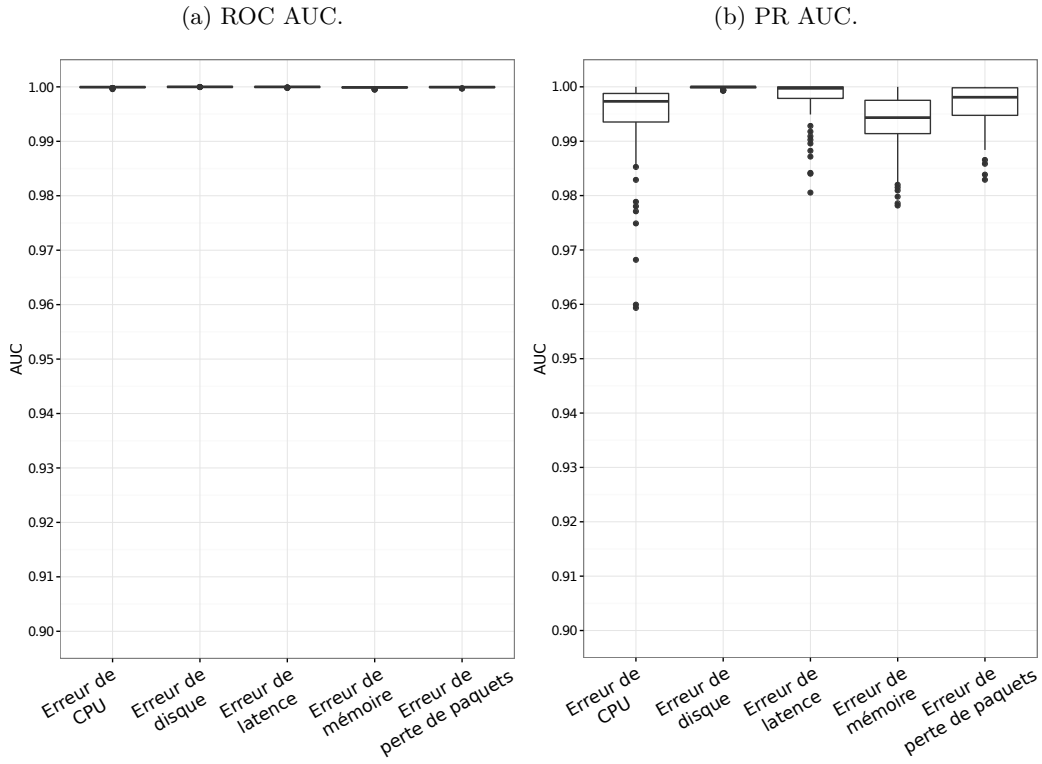


FIGURE 5.2.3 – Détection d'erreurs selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{Y}$ en *Detect_Sup*.

une PR AUC moyenne de 0.73. Le sous-flux réseau quant à lui détecte très bien les erreurs de perte de paquets avec une ROC AUC moyenne de 0.99 et une PR AUC moyenne de 0.85.

Ainsi la détection d'une erreur n'est pas nécessairement la meilleure à partir d'un sous-flux dont les compteurs sont liés au type de l'erreur (l'erreur de latence est dans notre cas mieux détectée par le sous-flux mémoire et non pas par le sous-flux réseau).

Ce dernier point est lié à la nature de l'application qui est lancée sur les VMs. Dans le cas de MongoDB qui est un SGBD orienté mémoire (i.e. toutes les données sont mises en mémoire), l'activité mémoire est un témoin important de la bonne activité de l'application.

Les figures 5.3.2 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon l'erreur injectée par détection DESC hybride agrégée.

Toutes les erreurs sont détectées avec d'excellentes ROC AUC de moyenne 0.99, chacune par la détection sur au moins un sous-flux de données. Les erreurs de CPU, disque, mémoire et perte de paquets sont détectées avec une PR AUC moyenne excellentes (supérieure à 0.90). Les erreurs de latence réseau sont détectées avec une une PR AUC moyenne de 0.76 pour une performance acceptable.

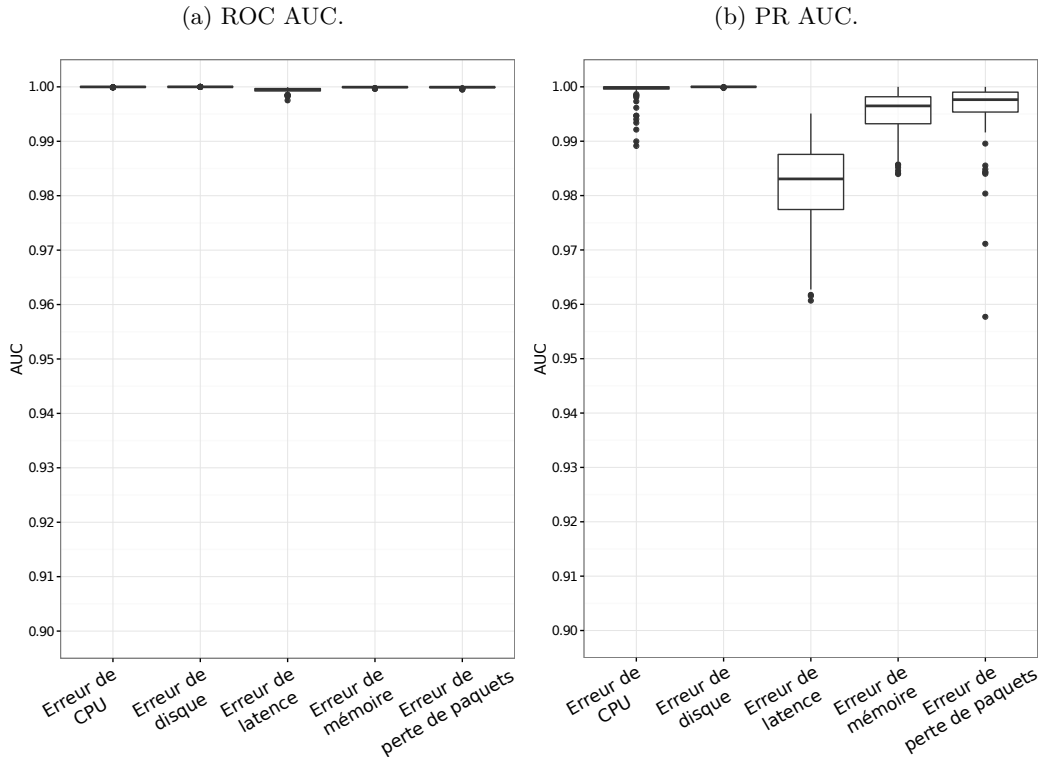


FIGURE 5.2.4 – Détection d'erreurs selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{Z}$ en *Detect_Sup*.

5.3.2 Durée de validité du jeu d'entrainement

Les figures 5.3.3 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon l'erreur locale à la VM observée par détection DESC hybride agrégée et avec un entrainement sur le jeu de données $\mathcal{Y}$ et une prédiction sur le jeu de données $\mathcal{Z}$. La prédiction a donc lieu sur des données collectées 5 mois après la collecte des données d'entrainement des modèles.

Les ROC AUC moyennes restent excellentes pour la détection des erreurs CPU, mémoire et perte de paquets, car proches de 0.99. Elles sont acceptables pour les erreurs de disque mais trop basses pour les erreurs de latence réseau (respectivement 0.75 et 0.41). La précision et le rappel sont eux grandement impactés par l'écart de temps entre les deux jeux de données.

Les figures 5.3.4 (a) et (b) présentent les ROC et PR AUC calculées pour la détection binaire d'erreurs à partir des descripteurs CS et DtR par détection DESC hybride agrégée et avec un entrainement sur le jeu de données $\mathcal{Y}$ et une prédiction sur le jeu de données $\mathcal{Z}$. La détection binaire apporte une PR AUC moyenne excellente et une ROC AUC acceptable pour notre contexte.

La détection d'erreurs selon le type d'erreur locale appliquée avec un entrainement sur le jeu de données $\mathcal{Y}$ et une prédiction sur le jeu de données $\mathcal{Z}$ ne fournit

(a) ROC AUC. (b) PR AUC.

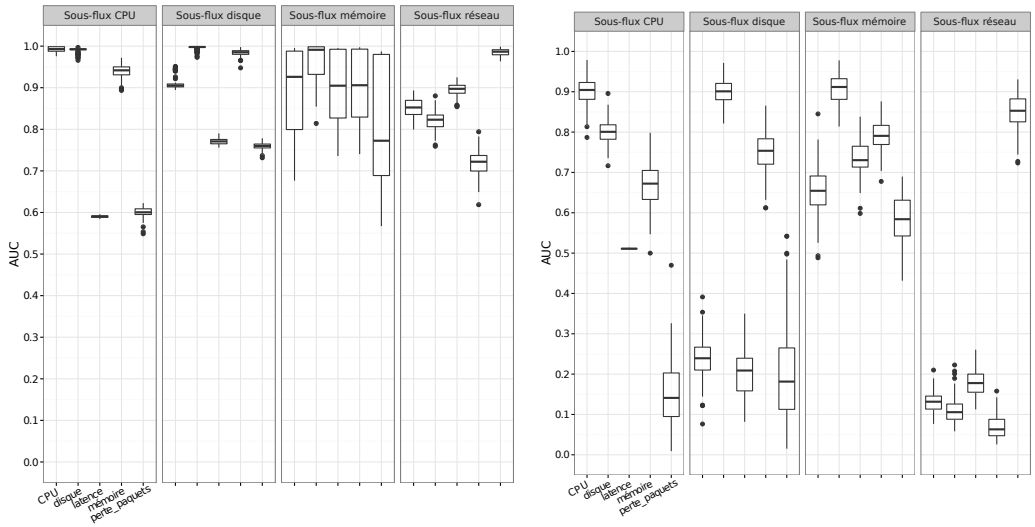


FIGURE 5.3.1 – Détection d'erreurs selon le type d'erreur locale à la VM observée en *Detect_DESC_hybride* par sous-flux.

(a) ROC AUC. (b) PR AUC.

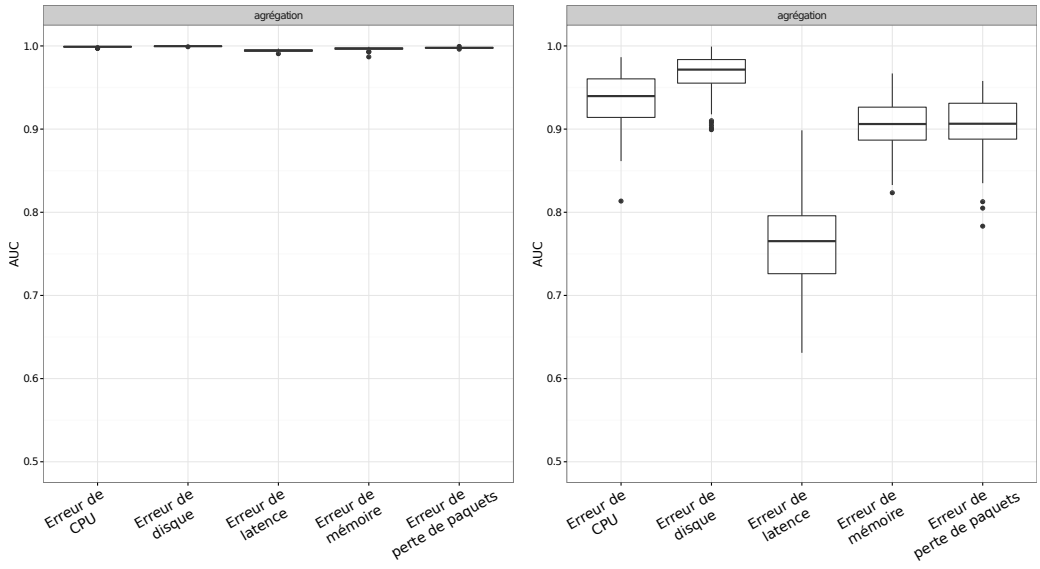


FIGURE 5.3.2 – Détection d'erreurs selon le type d'erreur locale à la VM observée, en *Detect_DESC_hybride* agrégée.

pas de résultats satisfaisants. Les résultats de classification binaire d'erreur appliquée sur les mêmes données sont quant à eux acceptables pour notre contexte.

(a) ROC AUC. (b) PR AUC.

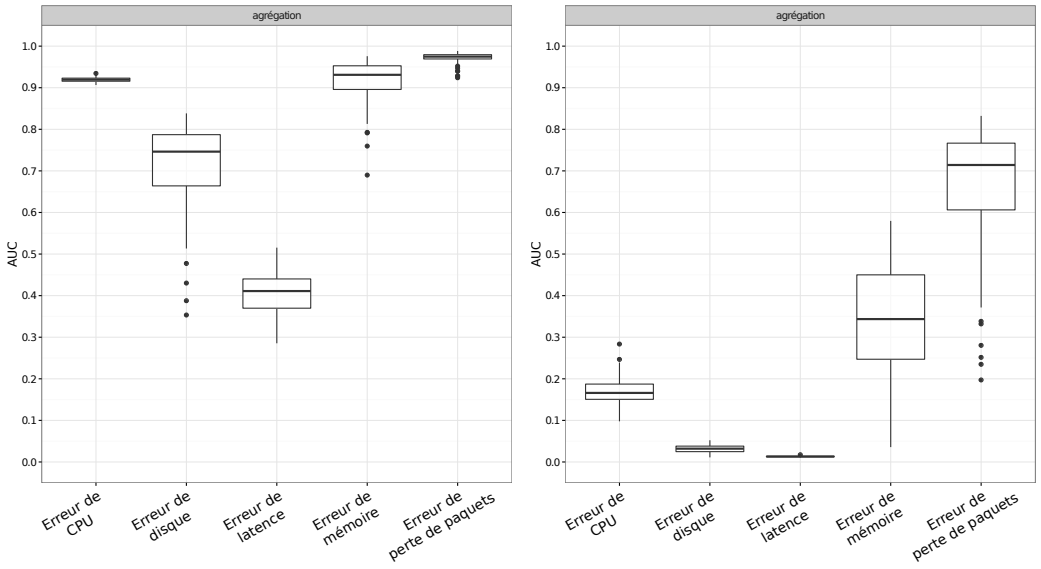


FIGURE 5.3.3 – Détection d'erreurs selon le type d'erreur locale à la VM observée, avec un apprentissage sur $\mathcal{Y}$ et une prédiction sur $\mathcal{Z}$, en *Detect_DESC_hybride* agrégée.

(a) ROC AUC. (b) PR AUC.

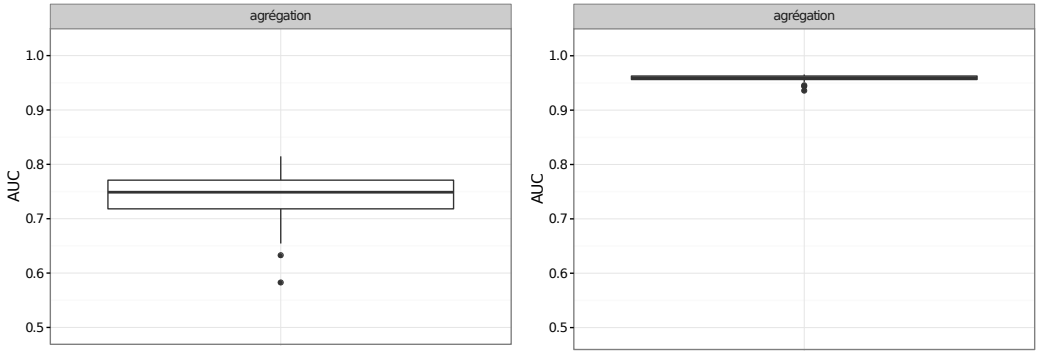


FIGURE 5.3.4 – Détection binaire d'erreur, avec un apprentissage sur $\mathcal{Y}$ et une prédiction sur $\mathcal{Z}$, en *Detect_DESC_hybride* agrégée.

Les deux tests montrent que l'apprentissage de notre modèle de classification Random Forest n'est plus efficace au bout de 5 mois d'activité.
La détection DESC hybride amène une meilleure PR AUC moyenne et une ROC AUC moyenne légèrement inférieure pour une détection binaire et un entrainement sur le jeu de données $\mathcal{Y}$ et une prédiction sur le jeu de données $\mathcal{Z}$, que le même test avec la détection par apprentissage supervisé implémentée avec Random Forest (voir figures 5.2.2). Ces résultats sont très prometteurs pour notre détection DESC hybride.

Ce problème de jeu d'entrainement non actualisé peut notamment être géré par la mise en place d'un algorithme Random Forest à apprentissage semi supervisé comme l'algorithme présenté dans [Leistner 2009]. Ainsi, les erreurs constatées par les administrateurs des services cloud ou bien par leurs utilisateurs peuvent être fournies à l'algorithme en ligne afin d'actualiser ses arbres de décision au fur et à mesure.

Lors de l'évaluation dans le chapitre 6 de notre deuxième cas d'étude Clearwater, une période de 1 mois entre la collecte de deux jeux de données est notamment testée afin de vérifier à quelle période un modèle n'est plus pertinent.

5.4 Analyse des compteurs de performance utiles à la détection

Nous avons collecté les 10 compteurs de performance les plus utilisés dans les arbres d'un modèle construit sur l'algorithme Random Forest pour la détection multi-classe selon le type d'erreur locale à une VM observée. Ces compteurs sont présentés dans le tableau 5.2. Le jeu de données $\mathcal{Z}$ est utilisé pour obtenir les résultats suivants.

L'analyse des compteurs de performance grâce au code couleur présenté dans la section 4.6 nous permet de constater plusieurs points qui sont les suivants :

- Les compteurs réseau sont majoritaires pour la détection d'erreurs de perte de paquets.
- Les compteurs mémoire sont majoritaires pour la détection d'erreurs de latence réseau (ceci est cohérent avec le fait que la détection DESC hybride détecte mieux les erreurs de latence avec le sous-flux mémoire, voir la sous section 5.3.1). Seul un compteur de performance réseau est compté pour cette détection. Après analyse, nous expliquons cela par le fait qu'une injection de latence réseau amène la VM primaire observée à perdre son rôle de VM primaire dynamiquement de par son temps de réponse trop long. Ainsi, le rôle de la VM change et c'est surtout son activité mémoire qui témoigne de ce changement de rôle induit par notre injection de latence.
- Les compteurs mémoire sont majoritaires pour la détection d'erreurs de mémoire.
- Les compteurs CPU sont majoritaires pour la détection d'erreurs de CPU.
- Les compteurs disque sont majoritaires pour la détection d'erreurs de disque. Ils sont très présents pour la détection de la majorité des erreurs (CPU, mémoire, disque et latence réseau). Cela est dû au fait que MongoDB est un SGBD orienté mémoire. MongoDB charge dans toute la mémoire libre d'une machine les documents qui ont besoin d'être stockés dans cette machine. Peu d'accès disque sont donc fait usuellement. Les injections réalisées demandent en majorité davantage de mémoire pour être exécutées et amènent la machine à réduire l'espace mémoire disponible pour la BDD et donc à augmenter les accès disque de l'application MongoDB. Dès qu'un accès disque est comp-

tabilisé ou que MongoDB ne monopolise pas la majorité de la mémoire, les modèles de détection repèrent un comportement anormal.

Ainsi nous constatons que les compteurs liés au type d'une erreur (par exemple les compteurs du groupe CPU sont liés à l'erreur de CPU) aident majoritairement la détection et le diagnostic de cette erreur. Cela n'est toutefois pas valable pour les injections de latence réseau qui sont majoritairement détectées grâce à des compteurs mémoire.

Cette analyse de compteurs utiles pour la détection permet de visualiser les données de monitoring stratégiques nécessaires à la détection et par là même montre que ces données dépendent des erreurs détectées. Ces compteurs semblent de plus dépendre de l'implémentation du système cible. Cette étude est donc approfondie pour le deuxième cas d'étude dans la sous section 6.4 afin d'obtenir une confirmation.

5.5 Conclusion

Dans ce chapitre nous avons testé les performances de détection de notre stratégie de détection d'anoamalie. Plusieurs algorithmes à apprentissage supervisé ont été testés de même que la détection DESC hybride par sous-flux et agrégée. Les détections binaires et avec diagnostic sont menées sur deux cas : 1) le test fait sur un unique jeu de données (divisé en deux parties d'apprentissage et de prédiction), 2) le test avec deux jeux de données, un pour l'entrainement de modèles et l'autre pour la prédiction avec la collecte du jeu de données pour la prédiction réalisée 5 mois après la collectée de données d'entrainement. Les performances de détection de Random forests sur les autres algorithmes testés sont supérieures dans les deux cas.

Pour une détection par apprentissage supervisé avec l'algorithme Random Forest ou une détection par détection DESC hybride, les performances de détection avec diagnostic sont bonnes dans le premier cas mais ne sont pas satisfaisantes dans le second cas.

Dans le cas binaire, les performances de détection sont acceptables pour une détection par apprentissage supervisé avec Random Forest ou une détection par détection DESC hybride et avec des performances comparables.

Concernant les compteurs de performance utiles à la détection d'anomalies, nous constatons que les compteurs liés au type d'une erreur aident majoritairement la détection et le diagnostic de cette erreur. Cela n'est toutefois pas valable pour les injections de latence réseau qui sont majoritairement détectées grâce à des compteurs mémoire. En effet, ces injections amènent la VM primaire observée à perdre son rôle et à devenir un replica. Ainsi, nous remarquons que c'est surtout son activité mémoire qui témoigne de ce changement de rôle induit par notre erreur de latence. Nous concluons donc que les compteurs utiles à la détection d'une erreur pourraient être dépendants du cas d'étude. Cette même analyse a de plus pu être faite avec l'étude de détection par détection DESC hybride par sous-flux.

Ce point sera confirmé dans le chapitre suivant traitant le cas d'étude Clear-

Classe d'erreur	Détection du type d'erreur locale à l'origine d'une erreur
Perte de paquets	tcpext_tcpsackfailures tcp_retrans_percentage mem_mapped tx_bytes_lo tcp_retranssegs rx_bytes_lo tcp_attemptfails rx_pkts_lo tcpext_tcptimeouts tx_pkts_lo
Latence réseau	mem_mapped proc_total swap_free mem_cached disk_free_percent_rootfs mem_free disk_free mem_buffers disk_free_absolute_rootfs tx_bytes_eth0
Mémoire	mem_cached diskstat_sda_read_bytes_per_sec io_reads diskstat_sda_reads mem_buffers vm_pgpgin io_nread vm_pgmajfault swap_free mem_free
Disque	load_one cpu_idle proc_total diskstat_sda_write_bytes_per_sec vm_pgpgout mem_dirty disk_free io_writes diskstat_sda_weighted_io_time part_max_used

Classe d'erreur	Détection du type d'erreur locale à l'origine d'une erreur
CPU	cpu_user cpu_idle load_one proc_run load_five proc_total swap_free mem_mapped disk_free_percent_rootfs load_fifteen

Tableau 5.2 – Compteurs les plus utiles en monitoring hyperviseur pour une détection multi-classe selon le type d'erreur locale à la VM observée

water. Ce chapitre présentera également des tests de détection approfondis afin d'évaluer notre stratégie de détection sur un cas d'étude plus complexe.

Chapitre 6

Évaluation du cas d'étude VNF : Clearwater

Sommaire

6.1 Présentation **94**
6.1.1 Description et déploiement 94
6.1.2 Charge de travail 95
6.1.3 Jeux de données 95
6.2 Évaluation de la détection par apprentissage supervisé **97**
6.2.1 Détection d'erreurs 97
6.2.2 Détection de symptômes préliminaires à une violation de service 107
6.2.3 Détection de violations de service 109
6.3 Évaluation de la détection DESC hybride **114**
6.3.1 Détection d'erreurs 115
6.3.2 Détection de violations de service 117
6.3.3 Durée de validité du jeu d'entrainement 117
6.4 Analyse des compteurs de performance utiles à la détection **119**
6.5 Conclusion **128**

Ce chapitre présente l'évaluation expérimentale de notre stratégie de détection d'anomalies.

L'évaluation présentée approfondit l'évaluation menée dans le chapite 5. Elle recense plusieurs tests de performances effectués sur une VNF Clearwater [1]. Clearwater est un IMS représentatif des télécommunications et spécialement implémenté pour les clouds. Il constitue un cas d'étude plus complexe que le cas de MongoDB de par la diversité des rôles de ses composants applicatifs (par exemple proxy, routeur et BDD). Ce cas d'étude a été réalisé en collaboration avec Orange Labs [Sauvanaud 2016].

1. Un réseau composé de VNFs associées à une infrastructure virtualisée ainsi qu'une couche de management du cycle de vie des VNFs, constitue une architecture de virtualisation de fonctions réseau, notée NFV pour Network Function Virtualization [ETSI Group Speciafication NFV 2013].

6.1 Présentation

6.1.1 Description et déploiement

Clearwater permet de passer des appels audio et vidéo et offre un service de messagerie par le biais du protocole SIP (Session Initiation Protocol). Cet IMS implémente des fonctions avec leurs interfaces normalisées correspondantes. Il n'implémente toutefois pas de cœur de réseau. Il comprend six composants appelés Bono, Sprout, Homestead, Homer, Ralf, et Ellis (représentés figure 6.1.1).

Bono est le proxy SIP qui implémente la fonction de serveur mandataire du contrôle d'appel ou Proxy-Call/Session Control Functions (P-CSCF). Il est le point de contact de l'IMS avec les utilisateurs et route leurs requêtes SIP vers Sprout. Il gère certaines données de facturation qu'il transmet à Ralf. Bono gère aussi la translation d'adresse (en anglais Network Address Translation ou NAT).

Sprout est le routeur SIP qui reçoit les requêtes de Bono et les route aux services correspondants. Il implémente la fonction Interrogating-CSCF (I-CSCF) qui interroge un Home Suscriber Server (HSS) afin d'assigner un Serving-CSCF (S-CSCF) à un utilisateur. Sprout implémente aussi la fonction S-CSCF qui gère l'interaction avec les serveurs d'application. Sprout contient lui-même un serveur de téléphonie multimédia (MMTel), dont les données sont stockées dans Homer.

Homestead est un serveur HTTP RESTful (souvent abrégé "Hs" dans nos figures et tableaux). Deux cas de figure sont envisageables. Homestead peut soit stocker et gérer les données liées aux abonnements des utilisateurs dans une base de données Cassandra, soit récupérer les données d'un HSS tiers compatible IMS. La tâche d'un HSS miroir est comprise dans les fonctions I-CSCF et S-CSCF.

Ainsi, Bono, Sprout et Homestead implémentent ensemble l'intégralité du contrôle d'appel (CSCF).

Homer est un serveur de gestion de documents (en anglais XML Document Management Serveur ou XDMS), et s'appuie sur une base de données Cassandra. Homer stocke les configurations MMtel des utilisateurs de l'IMS.

Ralf implémente la fonction de facturation appelée Charging Trigger Function (CTF). Il facture les communications enregistrées par Bono et Sprout et les envoie à un serveur Charging Data Function (CDF). Ce serveur n'est pas implémenté dans Clearwater et Ralf ne fonctionne que s'il a accès à un serveur CDF tiers.

Ellis est un portail web de création d'utilisateurs. Il est utilisé seulement dans le but de tester un déploiement Clearwater.

Clearwater gère la montée en charge grâce à une répartition de charge DNS (pour Domain Name System). Notre déploiement comprend Bono, Sprout, Homestead et Homer. Chacun de ces composants est déployé sur une VM (voir Figure 6.1.1). Nous ne configurons pas la fonction de facturation. De plus, nous ne réalisons pas un déploiement redondant afin d'évaluer simplement l'impact de nos injections sans faire d'hypothèse sur les effets de la redondance (la redondance de notre SGBD Mongo était simple à interpréter, ici nous aurions à prendre en compte la redondance de trois applications complexes). La redondance peut en effet permettre d'éviter

des anomalies juste après leur occurence. De cette manière l'évaluation de notre stratégie de détection pourrait être imcomplète en ce que la stratégie ne pourrait jamais être testée pour la détection de ces anomalies évitées.

Nous focalisons notre étude sur les composants Bono, Sprout et Homestead qui assurent le contrôle d'appel. Les campagnes d'injection ciblent ces trois VMs. Nous donnons de plus les résultats de performance pour ces trois VMs qui constitue notre système cible.

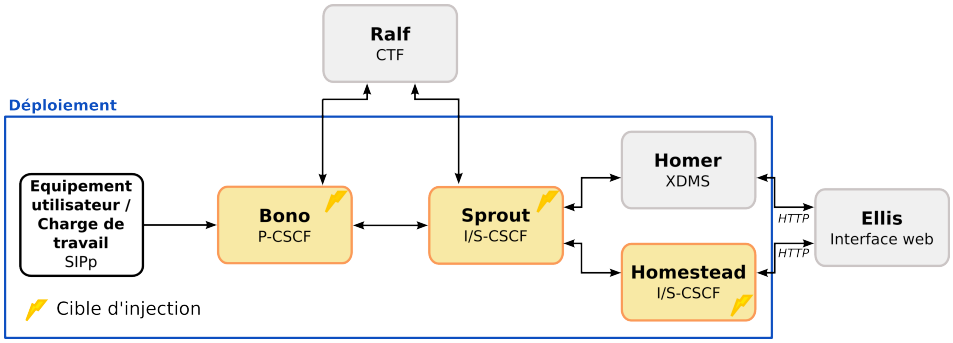


FIGURE 6.1.1 – Déploiement Clearwater.

6.1.2 Charge de travail

Nous utilisons le benchmark SIPp [2] afin de simuler les conditions opérationnelles de Clearwater. Le benchmark est configuré pour simuler des utilisateurs qui procèdent chacun à un appel à un autre utilisateur. Chaque appel est constitué de messages REGISTER, INVITE, UPDATE et BYE. L'organisation des messages d'un appel est décrite dans un scénario XML donné en annexe B. Un compteur de 10 secondes est positionné pour l'envoi et la réception de messages afin d'invalider une action longue à l'exécution. Le benchmark est configuré avec un nombre fixe d'appels pouvant être réalisé par seconde.

6.1.3 Jeux de données

6.1.3.1 Description

Nous avons effectué quatre expérimentations qui nous ont permis de collecter quatre jeux de données appelés $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, et $\mathcal{D}$. Chacun a été collecté lors de l'exécution d'une campagne d'injection à partir de notre plateforme Clearwater. Pour chaque campagne, nous avons les paramètres communs suivants :

— $l_vm = \{Bono, Sprout, Homestead\}$.
— $l_erreur = \{CPU, mémoire, disque, latence, perte_paquet\}$,
— $pause = 100$ min.
— La collecte des données est faite par monitoring SE et hyperviseur.

2. http ://sipp.sourceforge.net/index.html (vu octobre 2016)

La différence entre les jeux de données est présentée ci-dessous.

— Jeu de données $\mathcal{A}$:
  — $l_intensité = \{1 : 7\}$
  — $d_injection = 10$ min,
  — $\mathcal{A}$ contient environ 46600 observations pour chaque VM (environ 10 jours de monitoring).
— Jeu de données $\mathcal{B}$:
  — $l_intensité = \{4, 7\}$
  — $d_injection = 10$ min,
  — $\mathcal{B}$ a été enregistré lors d'une campagne d'injection lancée 1 mois après la fin de la campagne de $\mathcal{A}$. Entre la collecte de $\mathcal{A}$ et de $\mathcal{B}$, nous avons utilisé notre plateforme pour d'autres tests qui ne sont pas précisés dans cette thèse. Aucune mise à jour logicielle n'a été effectuée.
  — $\mathcal{B}$ contient environ 13600 observations pour chaque VM (environ 2 jours et demi de monitoring).
— Jeu de données $\mathcal{C}$:
  — $l_intensité = \{4, 7\}$,
  — $d_injection = 4$ min,
  — $\mathcal{C}$ a été enregistré lors d'une campagne d'injection lancée juste après la fin de campagne de $\mathcal{B}$.
  — $\mathcal{C}$ contient environ 12800 observations pour chaque VM.

  Jeu de données $\mathcal{D}$:
  — $l_intensité = \{4, 7\}$,
  — $d_injection = 10$ min,
  — $\mathcal{D}$ a été enregistré lors d'une campagne d'injection lancée 1 semaine après la fin de campagne de $\mathcal{B}$.

Nous faisons les commentaires suivants sur le choix des configurations de nos expérimentations :

— Ces trois expérimentations nous permettent d'évaluer les performances de notre stratégie de détection pour des prédictions menées 1 mois (en comparant $\mathcal{A}$ et $\mathcal{B}$) ainsi que 1 semaine (en comparant $\mathcal{B}$ et $\mathcal{D}$) après l'entrainement de modèles.
— Seules deux intensités d'injection sont conservées dans les campagnes d'injection de $\mathcal{B}$ et $\mathcal{C}$ afin d'évaluer les performances de détection DESC sur ces deux seules intensités. Nous voulons ici tester si nos résultats sont meilleurs avec des erreurs injectées d'intensité supérieure à 4.
— Lorsque les jeux de données sont comparés entre eux, seules les intensités d'injection 4 et 7 sont conservées dans $\mathcal{A}$. Les autres intensités sont conservées dans les cas où l'on souhaite tester si un jeu de données plus détaillé peut améliorer les performances de détection.

6.1.3.2 Utilisation

Dans la présentation des résultats qui suit, lorsqu'il n'est pas fait mention du jeu de données utilisé lors d'un test, le jeu de données $\mathcal{A}$ est utilisé en monitoring hyperviseur.

De plus, la limite PE_max pour la détection de symptômes préliminaires à une violation de service ou de violation de service est fixée à 2%. Nous rappelons que PE_max correspond au pourcentage maximal acceptable de requêtes non traitées avec succès par minute par le système (voir la sous section 3.3.1 du chapitre 3).

6.2 Évaluation de la détection par apprentissage supervisé

Cette section présente les tests menés pour la détection d'anomalies en utilisant la détection par apprentissage supervisé appliquée avec l'algorithme Random Forest (*Detect_Sup_RF*). Ces tests sont réalisés sur nos trois niveaux de détection ainsi que nos trois types de détection (le détail des niveaux et types de détection est présenté dans le chapitre 3 section 3.1). Ces tests sont menés en appliquant une détection sur les données de VMs observées Bono, Homestead (noté Hs) ou Sprout.

6.2.1 Détection d'erreurs

La détection d'erreurs a été traitée grâce à plusieurs tests. Des études de sensibilité sont menées sur la distribution des données, la taille du jeu de données et la durée d'injection. Nous présentons de plus les résultats de détection d'erreurs avec le but de diagnostiquer (en terme plus technique c'est une détection multi-classe) le type d'erreur et la VM à l'origine d'une erreur. Enfin nous évaluons si un modèle est toujours valide au bout de 1 mois et 1 semaine d'activité du système cible.

6.2.1.1 Sensibilité à la distribution des données

Comme expliqué dans le chapitre 2, la distribution des classes dans un jeu de données influence les résultats. Nous présentons ici l'impact d'un changement de distribution sur nos données. L'évaluation utilise le jeu de données $\mathcal{A}$ sous-échantillonné aléatoirement de manière à contenir un nombre d'observations d'erreurs qui correspond à 2% du nombre d'observations de comportement normal (ce qui revient à environ 1000 observations d'erreur tout type d'erreur confondu). Le pourcentage d'erreurs de notre jeu de données est augmenté à 5%, 10% et 50% en sous-échantillonnant les observations de comportement normal.

Les figures 6.2.1 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs binaire dans Bono et ce, pour chaque distribution (soit 2%, 5%, 10% et 50%). Elles montrent que dans ce cas, l'augmentation de PR AUC est en moyenne de 0.02 pour les distributions entre 2% et 50% d'erreur. Comme attendu, les ROC AUC restent stables.

Cet écart est très faible car les résultats sont déjà excellents en considérant la distribution de 2%. Une campagne d'injection durant 10 jours (en considérant un sous-échantillonnage du jeu de données associé dans le but d'obtenir une distribution de 2%) est donc suffisante pour décrire le comportement normal du système lors de l'expérimentation de $\mathcal{A}$ et pour décrire les erreurs considérées dans notre modèle.

(a) ROC AUC. (b) PR AUC.

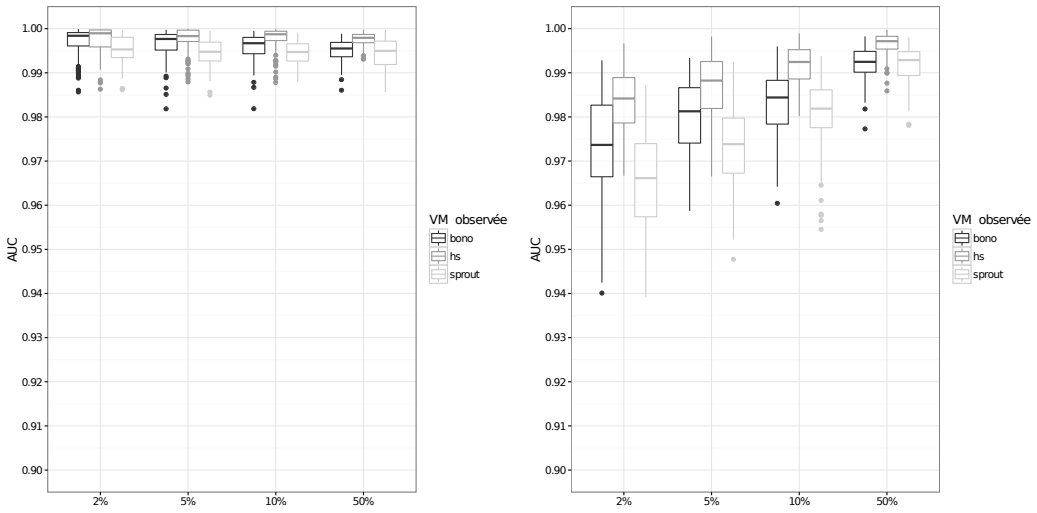


FIGURE 6.2.1 – Détection binaire d'erreurs avec plusieurs distributions en *Detect_Sup_RF*.

6.2.1.2 Détection avec diagnostic de types d'erreur

Les figures 6.2.2 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon le type d'erreur locale à la VM observée. Les résultats sont globalement satisfaisants avec des moyennes supérieures à 0.90, exception faite des cas cités ci-dessous.

L'erreur de latence réseau est détectée avec la moins bonne performance dans les VMs Sprout et Homestead avec une ROC AUC moyenne entre 0.82 et 0.85 et une PR AUC également entre 0.82 et 0.85. Une erreur de latence réseau dans la VM Bono qui a le rôle de proxy est par contre bien détectée. Ceci est dû à l'impact plus important qu'a le proxy sur toute la chaine de gestion de requêtes (une baisse de latence coupe en effet toute communication avec Sprout). L'erreur de perte de paquets est moins bien détectée lorsqu'elle est injectée dans Sprout avec des ROC et PR AUC moyennes de 0.89. Le rôle de Sprout dans l'IMS ne nous permet pas d'évoquer une raison pour cette différence de performance pour cette erreur. La raison doit être liée à des détails d'implémentation de la fonction

de routage hébergée par Sprout qui masque l'effet des erreurs sur les données de monitoring.

Nous pouvons conclure que selon le code exécuté par une VM pour accomplir sa fonction, les erreurs sont plus ou moins difficiles à détecter par les algorithmes (nous avions déjà constaté que l'impact d'injections de même intensité n'est pas le même selon la VM, voir les figures 4.4.2).
Enfin nous constatons ici qu'une détection avec diagnostic sur le type d'erreur locale à la VM observée (voir figure 6.2.2) baisse les performances par rapport à une détection binaire (voir les figures 6.2.1). Néanmoins les résultats restent élevés.

(a) ROC AUC. (b) PR AUC.

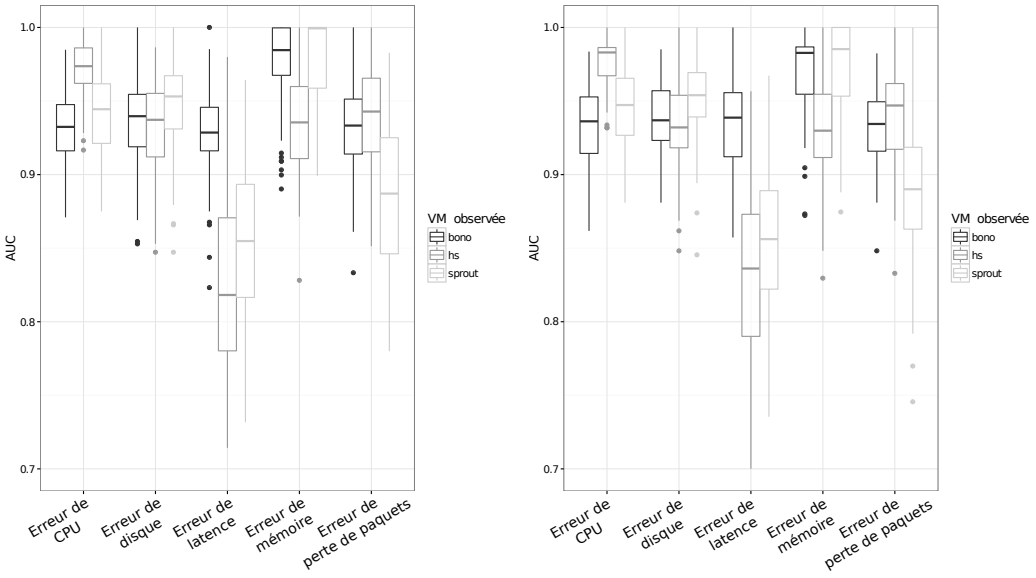


FIGURE 6.2.2 – Détection d'erreurs selon le type d'erreur locale à la VM observée en *Detect_Sup_RF*.

6.2.1.3 Détection avec diagnostic de la VM à l'origine d'erreur

Les figures 6.2.3 (a) et (b) présentent les ROC et PR AUC calculées pour la détection selon la VM à l'origine de l'erreur.

Les ROC AUC ont une moyenne de 0.99 et les PR AUC sont en moyenne supérieures à 0.95 pour Sprout et Homestead et supérieures à 0.91 pour Bono.

La détection selon la VM à l'origine de l'erreur est donc excellente. Nous constatons de plus ici que cette détection donne des performances de détection équivalentes au cas binaire.

En effet, des ROC AUC moyennes excellentes sont conservées comme celles de la détection binaire (voir figure 6.2.1 (a)). Les PR AUC sont quant à elles équivalentes,

excepté pour la VM observée Bono. Depuis les données de monitoring de Bono, il est légèrement plus difficile d'identifier l'origine de l'erreur. Les erreurs injectées dans Sprout et Homestead n'ont pas un impact sur Bono aussi grand qu'une injection dans Bono provoque sur Sprout et Homestead. La détection par Bono a tout de même des performances supérieures à 0.90 toutes classes confondues.

Nous constatons également que les meilleures performances de détection sont obtenues à partir des données observées dans Homestead quelle que soit l'origine de l'erreur.

Ainsi, nous pouvons conclure que la VM au début de la chaine CSCF (Bono, Sprout et Homestead composent dans cet ordre la chaine CSCF présentée figure 6.1.1) a donc la moins bonne visibilité du reste de la chaine et la VM à la fin de cette chaine possède la meilleure visibilité.

(a) ROC AUC. (b) PR AUC.

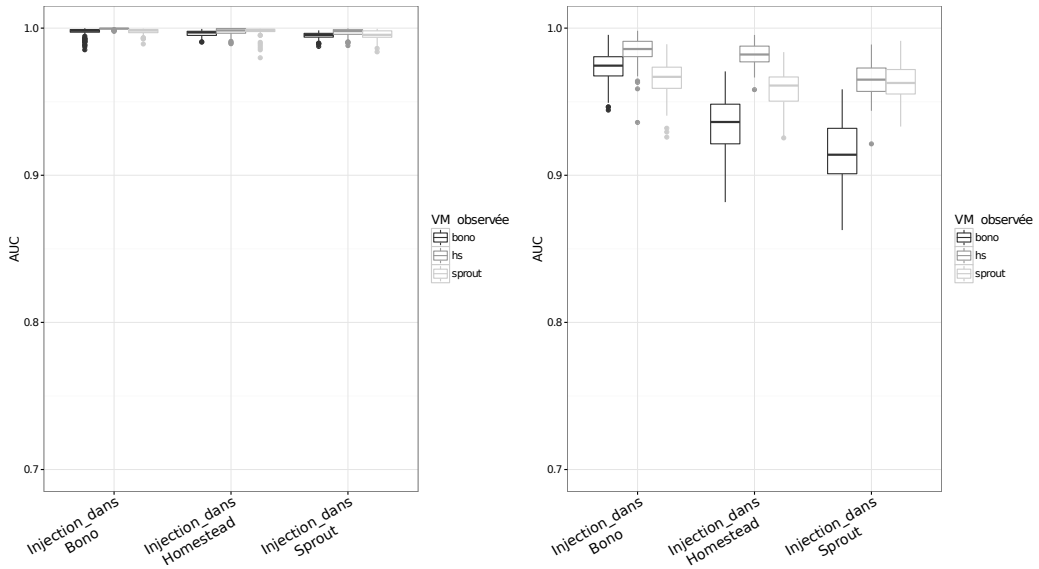


FIGURE 6.2.3 – Détection d'erreurs selon la VM à l'origine de l'erreur en *Detect_Sup_RF*.

6.2.1.4 Sensibilité aux intensités d'injection dans le jeu de données d'entrainement

Les figures 6.2.4 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon le type d'erreur locale à la VM observée pour un apprentissage sur le jeu de données $\mathcal{A}$ avec seulement les intensités 4 et 7 de chaque erreur représentée et une prédiction sur $\mathcal{A}$ avec toutes les intensités de chaque erreur représentée. Ces figures sont à comparer avec les figures 6.2.2 où toutes les intensités de $\mathcal{A}$ sont représentées dans le jeu de données d'entrainement.

Si l'on considère les ROC AUC, la détection des erreurs disque et mémoire est quasiment inchangée et reste excellente. La détection des erreurs CPU est par contre impactée par le manque d'exemples d'intensité dans le jeu de données d'entrainement mais reste acceptable dans notre contexte. Toutefois, les performances de détection des erreurs de latence réseau et de perte de paquets a beaucoup diminué. Elles restent acceptables pour Bono mais ne le sont plus pour Sprout et Homestead. Les PR AUC montrent la même tendance que les ROC AUC si ce n'est qu'elles sont très faibles pour la détection d'erreurs de latence réseau et de perte de paquets.

Nous constatons que les performances sont inférieures dans ce cas où l'entrainement est fait sur un sous-ensemble des intensités, surtout pour les VMs Homestead et Sprout qui ont dans ce cas des ROC AUC moyennes en dessous de 0.80 pour Sprout et Homestead.

(a) ROC AUC. (b) PR AUC.

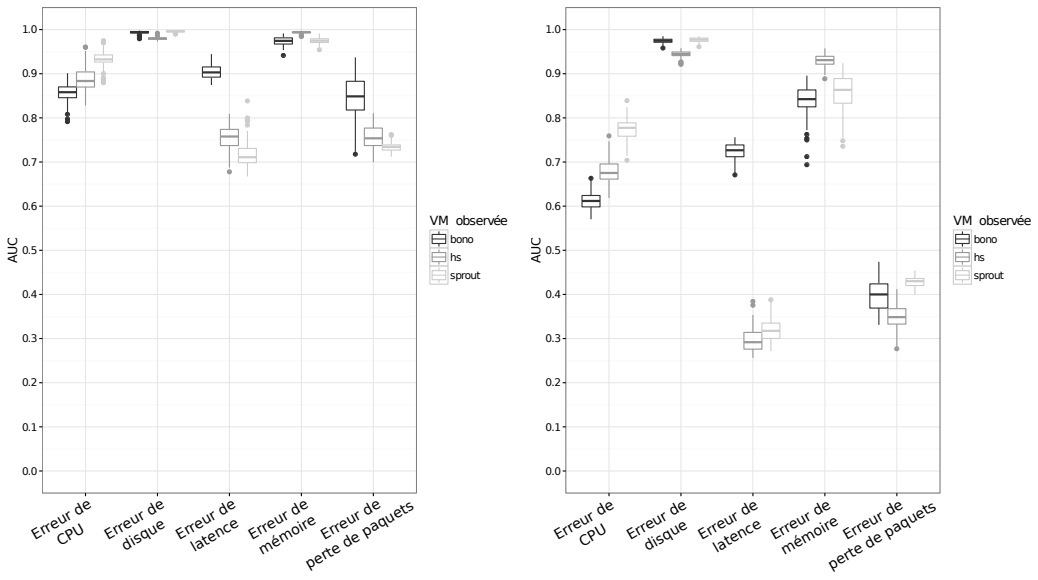


FIGURE 6.2.4 – Détection d'erreurs selon le type d'erreur locale à la VM observée avec un entrainement sur deux intensités de chaque erreur représentée dans $\mathcal{A}$ et une prédiction sur sept intensités de chaque erreur représentée dans $\mathcal{A}$ en *Detect_Sup_RF*.

6.2.1.5 Sensibilité à la taille du jeu de données

Cette étude de sensibilité à la taille du jeu de données est intéressante dans le contexte de l'étude de sensibilité suivante concernant la durée d'injection. Diminuer la durée d'injection revient en effet à diminuer le nombre d'observations d'erreurs collectées lors d'une campagne d'injection. La taille du jeu de données est donc

réduite.

Les figures 6.2.5 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon la VM à l'origine de l'erreur avec un sous-échantillonnage de $\mathcal{A}$; le nombre d'observations de comportement normal de $\mathcal{A}$ est réduit de 75%, le nombre d'observations d'erreur est réduit en conséquence de manière à représenter 2% du nombre d'observations de comportement normal.

Ces figures sont à comparer avec les figures 6.2.3 (a) et (b) qui présentent le même type d'analyse mais pour $\mathcal{A}$ sans sous-échantillonnage. Vis-à-vis des ROC AUC, les performances de détection sont, comme attendu, en moyenne similaires pour les deux cas comparés. Les performances sont toutefois inférieures si l'on considère les PR AUC moyennes. Elles varient entre 0.92 et 0.99 pour toutes les VMs confondues dans la figure 6.2.3 (b), mais varient entre 0.80 et 0.95 dans la figure 6.2.5 (b).

La taille du jeu de données d'entrainement a donc une influence importante dans notre détection. Dans notre test avec une réduction de 75% du jeu de données $\mathcal{A}$, les performances en terme de AUC varient avec une différence entre 0.05 et 0.10 si elles sont comparées avec les résultats en utilisant le jeu de données $\mathcal{A}$ avec 2% d'erreur mais sans réduction.

(a) ROC AUC. (b) PR AUC.

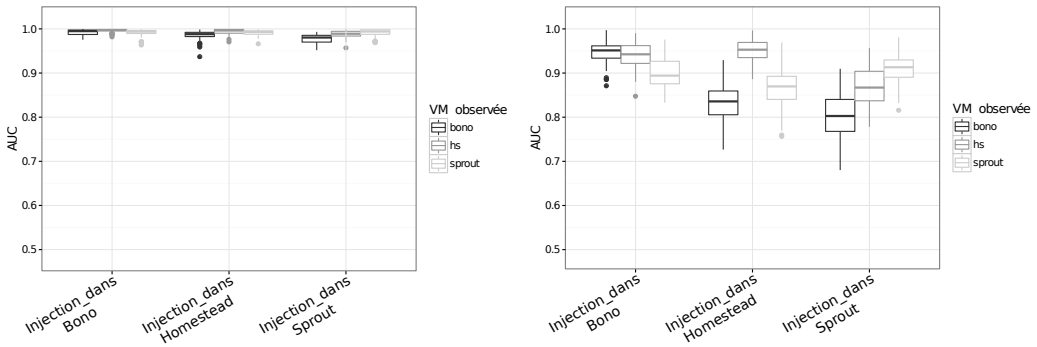


FIGURE 6.2.5 – Détection d'erreurs selon la VM à l'origine de l'injection avec le jeu de données $\mathcal{A}$ réduit de 75% en *Detect_Sup_RF*.

6.2.1.6 Sensibilité à la durée d'injection

Les figures 6.2.6 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon le type d'erreur locale à la VM observée, pour des injections durant 4 min (jeu de données $\mathcal{C}$). Ces figures sont à comparer avec les figures 6.2.2 (a) et (b) qui présentent le même type d'analyse mais pour des injections de 10 min. Les performances de détection sont inférieures à celles obtenues avec des injections de 10 min car les ROC AUC varient en moyenne entre 0.97 et 0.71 toutes VMs confondues.

(a) ROC AUC. (b) PR AUC.

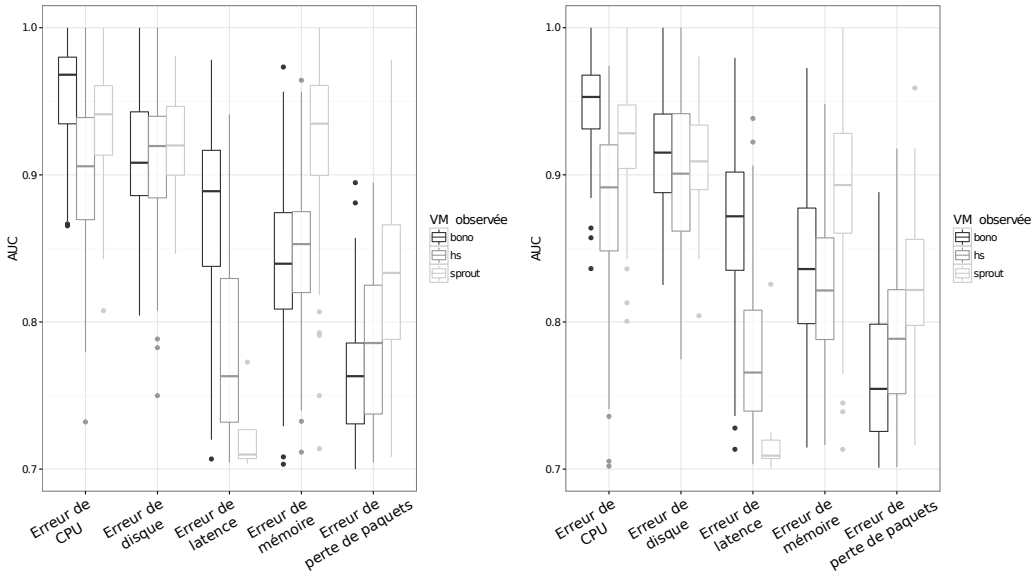


FIGURE 6.2.6 – Détection d'erreurs selon le type d'erreur locale à la VM observée et 4 min d'injection en *Detect_Sup_RF*.

(a) ROC AUC. (b) PR AUC.

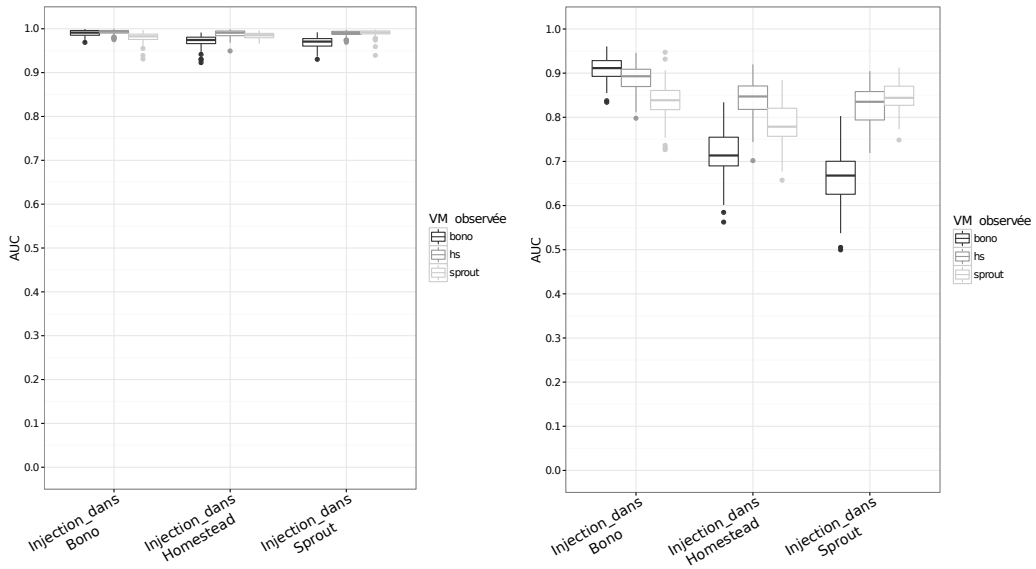


FIGURE 6.2.7 – Détection d'erreurs selon la VM à l'origine de l'erreur et 4 min d'injection en *Detect_Sup_RF*.

Afin d'aller plus loin dans la réflexion, les figures 6.2.7 (a) et (b) présentent les résultats calculés pour la détection de la VM à l'origine de l'erreur pour des injections durant 4 min. Ces figures sont à comparer avec les figures 6.2.3 (a) et (b) qui présentent le même type d'analyse mais pour des injections de 10 min. Les performances sont moins bonnes avec 4 min d'injection dans les deux cas de détection avec diagnostic du type d'erreur locale à la VM observée et selon la VM à l'origine de l'erreur. Cette différence est importante notamment pour les ROC AUC moyennes. Elle ne peut donc pas être due à une comparaison biaisée de deux jeux de données de taille différente car nous avons montré que lorsque la taille du jeu de données varie de 75% les ROC AUC moyennes sont similaires entre le plus petit jeu de données et le plus grand (voir les figures 6.2.5).

Une expérimentation avec des injections 20 min a de plus été menée. Les résultats sont similaires à ceux obtenus pour la détection avec des injections de 10 min.

Nous pouvons donc conclure que lors de l'évaluation d'une technique de détection avec un entrainement et une prédiction sur un même jeu de données, la durée d'injection peut influencer les résultats de détection d'erreurs. Des injections trop courtes par rapport à la période de monitoring (i.e., dans notre cas des injections de 4 min pour une collecte de 4 observations par minute) diminuent les performances de détection de manière importante.

6.2.1.7 Durée de validité du jeu d'entrainement

Les figures 6.2.8 (a) et (b) présentent les ROC et PR AUC calculées pour la détection binaire d'erreur, avec un entrainement sur le jeu de données $\mathcal{A}$ et une prédiction sur le jeu de données $\mathcal{B}$. La prédiction a donc lieu sur des données collectées 1 mois après la collecte des données d'entrainement du modèle.

Cette détection binaire donne des performances excellentes avec des ROC et PR AUC en moyenne supérieures à 0.90 pour ce cas de prédiction 1 mois après la collecte des données d'entrainement des modèles.

Les performances en détection selon le type d'erreur locale à la VM observée sont inférieures comme le montrent les figures 6.2.9 (a) et (b). Ces performances sont acceptables vis-à-vis des ROC AUC car celles-ci sont en moyenne supérieures à 0.85 excepté pour la détection de l'erreur de latence réseau qui est détectable avec des ROC AUC moyennes de 0.64, 0.78, et 0.89 pour Bono, Homestead et Sprout. Les PR AUC sont excellentes pour les erreurs CPU et disque et sont acceptables pour l'erreur de mémoire. Toutefois elles sont trop faibles pour la détection de latence réseau et de perte de paquets.

Nous constatons donc que dans le cas binaire, une prédiction qui a lieu 1 mois après l'entrainement d'un modèle n'est pas nécessaire. L'entrainement est toutefois nécessaire dans le cas de détection avec diagnostic.

Les figures 6.2.10 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon le type d'erreur locale à la VM observée, avec un entrainement sur le jeu de données $\mathcal{B}$ et une prédiction sur le jeu de données $\mathcal{D}$. La

(a) ROC AUC. (b) PR AUC.

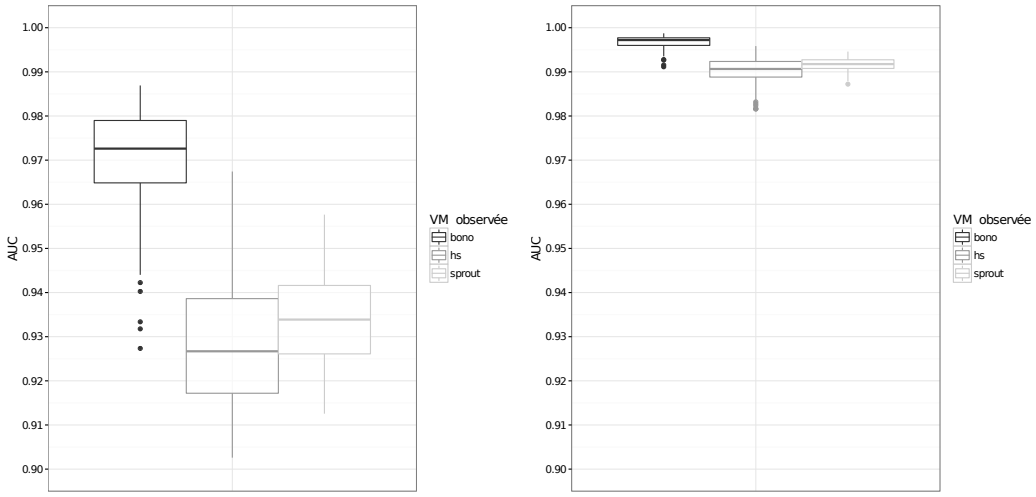


FIGURE 6.2.8 – Détection binaire d'erreurs avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*.

(a) ROC AUC. (b) PR AUC.

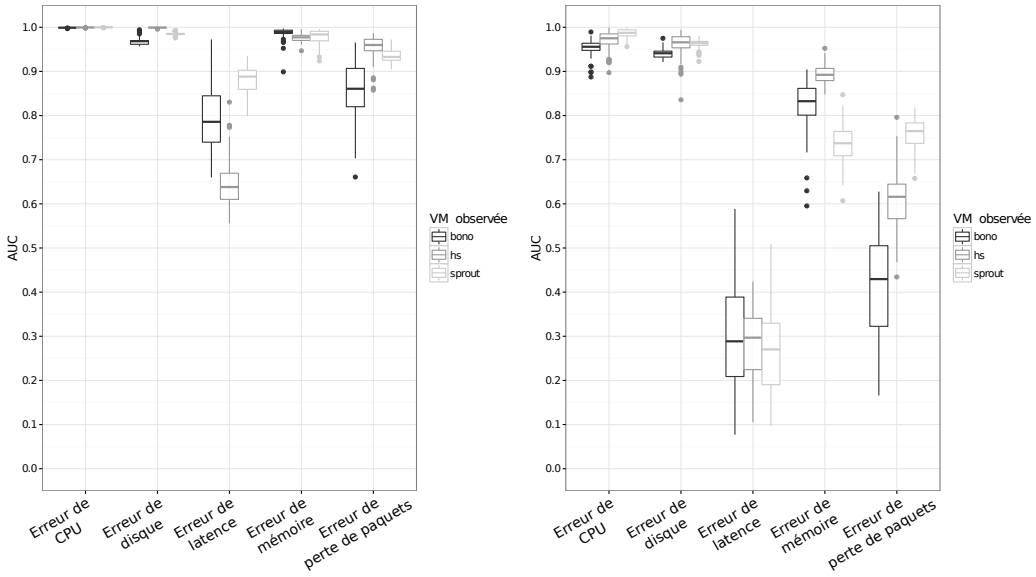


FIGURE 6.2.9 – Détection d'erreurs selon le type d'erreur locale à la VM observée avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*.

prédiction a donc lieu sur des données collectées 1 semaine après la collecte des données d'entrainement des modèles.

Les ROC AUC moyennes sont excellentes pour cette détection. Les PR AUC sont également excellentes pour la majorité des erreurs, exception faite des cas suivants. Les PR AUC moyennes montrent que l'erreur de latence est très mal détectée lorsqu'elle se produit dans Homestead et Sprout. L'erreur de mémoire est très mal détectée lorsqu'elle se produit dans Bono.

Les erreurs de latence sont très difficiles à détecter comme telles par un modèle au bout d'une semaine d'activité de ce modèle dans Sprout et Homestead.

(a) ROC AUC. (b) PR AUC.

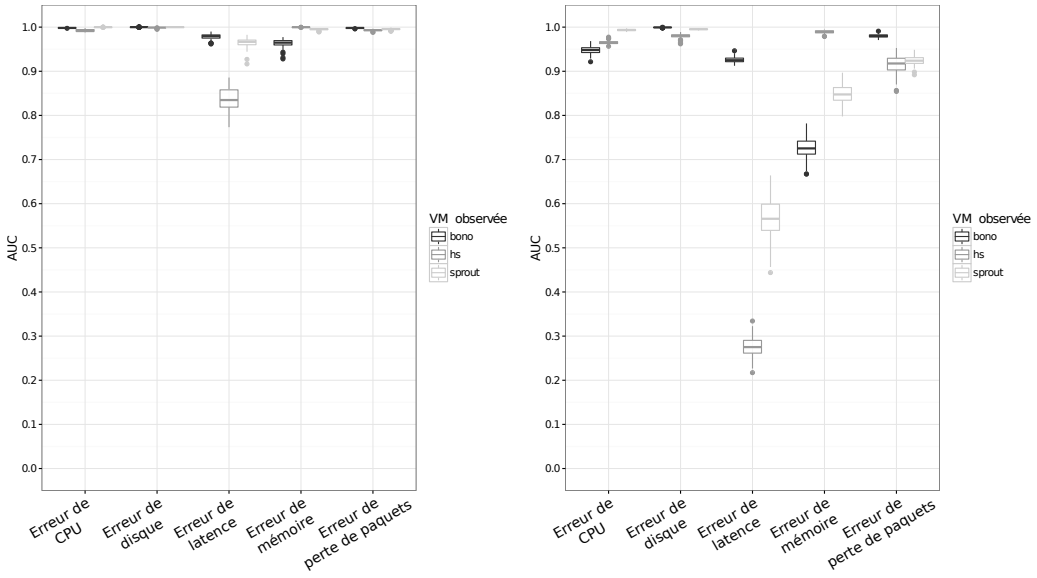


FIGURE 6.2.10 – Détection d'erreurs selon le type d'erreur locale à la VM observée avec un apprentissage sur $\mathcal{B}$ et une prédiction sur $\mathcal{D}$ (décalage de 1 semaine entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*.

Nous faisons de plus la comparaison avec les résultats obtenus à partir des jeux de données en monitoring SE. Les figures 6.2.11 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon l'erreur locale à la VM observée avec un entrainement sur le jeu de données $\mathcal{A}$ et une prédiction sur le jeu de données $\mathcal{B}$. La prédiction a donc lieu sur des données collectées 1 mois après la collecte des données d'entrainement du modèle.

Les performances sont bien meilleures que dans le cas du même test mais en monitoring hyperviseur (voir figure 6.2.9). Elles sont majoritairement excellentes vis-à-vis des ROC et PR AUC. Les valeurs les plus basses mais toujours acceptables, sont obtenues pour les erreurs de latence dans Sprout et Homestead. Dans ce cas de détection d'erreurs selon l'erreur locale à la VM observée et en monitoring SE, les modèles sont toujours pertinents pour la détection 1 mois après leur entrainement.

(a) ROC AUC. (b) PR AUC.

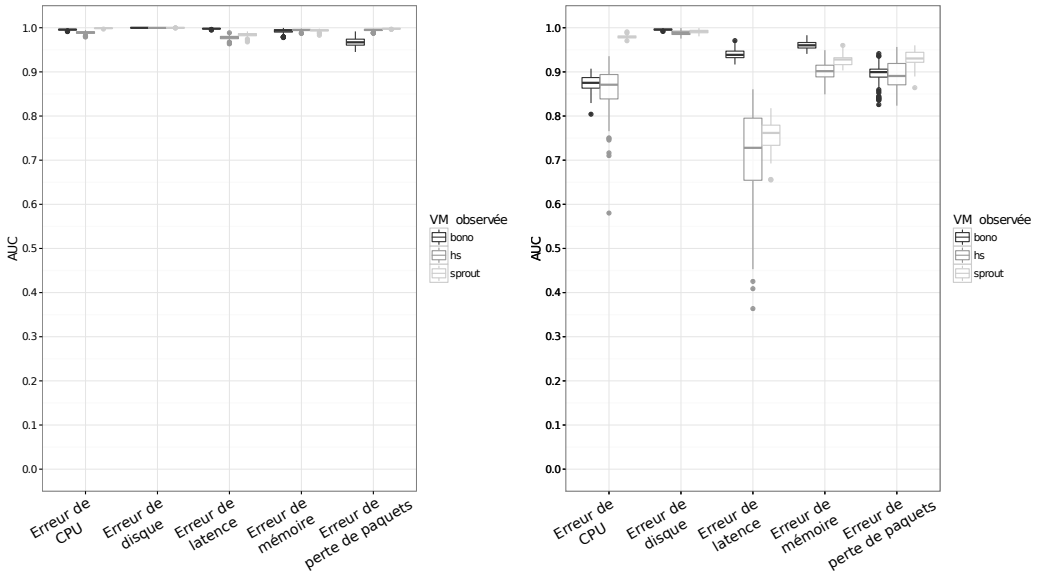


FIGURE 6.2.11 – Détection d'erreurs selon le type d'erreur locale à la VM observée en monitoring SE avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*.

6.2.2 Détection de symptômes préliminaires à une violation de service

La détection de symptômes préliminaires à une violation de service est utile car elle permet de déclencher des mesures de recouvrement avant qu'un utilisateur fasse l'expérience d'une baisse de service qui entraine une violation de service (voir chapitre 3).

Plusieurs tests sont menés afin d'évaluer les performances de détection associées :

— La détection est réalisée avec le diagnostic de la VM qui est à l'origine de la violation de service.

— Les performances pour une classification binaire et une prédiction sur des données collectées 1 semaine après la collecte des données d'entrainement

sont de plus présentées.

La détection de symptômes préliminaires à une violation de service ne permet pas de dissocier les alarmes selon le type d'erreur qui est à l'origine de la future violation de service. Les résultats correspondants ne sont donc pas présentés.

6.2.2.1 Détection avec diagnostic de la VM à l'origine d'une violation de service

Les figures 6.2.12 (a) et (b) présentent les ROC et PR AUC calculées pour la détection de symptômes préliminaires à des violations de service selon la VM à l'origine de la supposée violation de service. Les performances vis-à-vis des ROC AUC moyennes sont excellentes car elles sont proches de 0.98, toutes erreurs confondues. Les PR AUC moyennes sont excellentes pour Homestead car elles sont toujours supérieures à 0.90. Elles sont entre 0.88 et 0.89 pour Sprout et entre 0.73 et 0.84 pour Bono, ce qui est acceptable pour notre contexte mais un peu faible si l'on considère les performances pour Bono.

La détection de symptômes préliminaires à une violation de service est donc envisageable afin de détecter le type d'erreur présent dans une VM. Comme dans la partie 6.2.1.3, Homestead qui est la VM à la fin de la chaine CSCF est la plus performante pour l'identification de la localisation de la VM à l'origine d'une anomalie en cours.

(a) ROC AUC. (b) PR AUC.

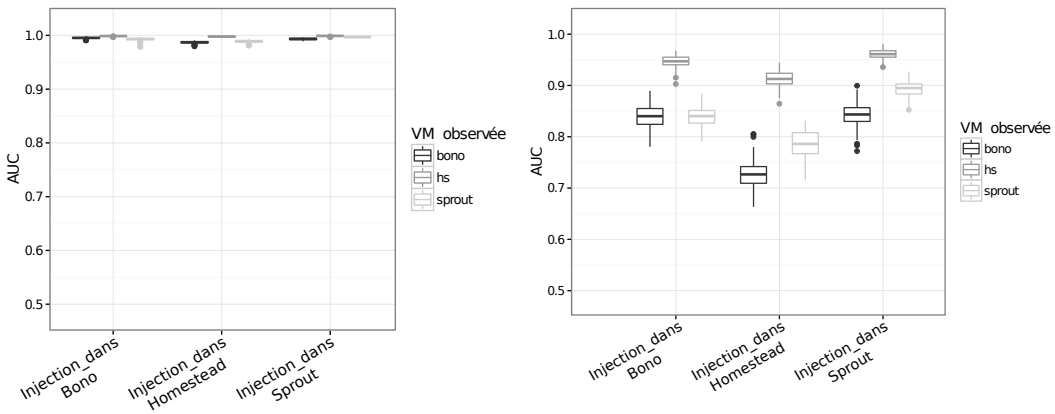


FIGURE 6.2.12 – Détection de symptômes préliminaires à des violations de service selon la VM à l'origine de la violation en *Detect_Sup_RF*.

6.2.2.2 Durée de validité du jeu d'entrainement

Les figures 6.2.13 (a) et (b) présentent les ROC et PR AUC calculées pour la détection binaire de symptômes préliminaires à une violation de service avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$. La prédiction a donc lieu sur des

données collectées 1 mois après la collecte des données d'entrainement des modèles. Nous montrons de plus les figures 6.2.14 (a) et (b) qui présentent les ROC et PR AUC calculées pour la détection binaire de symptômes préliminaires à une violation de service avec un apprentissage sur $\mathcal{B}$ et une prédiction sur $\mathcal{D}$. La prédiction a donc lieu dans ce second test sur des données collectées 1 semaine après la collecte des données d'entrainement des modèles. L'étude de détection avec diagnostic sur ces jeux de données n'est pas présentée car elle ne donne pas de performance acceptable dans notre contexte.

La détection binaire avec une prédiction 1 mois ou 1 semaine après la collecte des données d'entrainement des modèles offre une précision et un rappel excellents car les PR AUC moyennes sont supérieures à 0.90. Ce n'est toutefois pas le cas des ROC AUC moyennes. Trop de faux positifs sont détectés dès 1 semaine d'activité du modèle. Les performances obtenues en monitoring SE suivent la même tendance que les performances en monitoring hyperviseur. Elles sont légèrement supérieures mais ne sont pas acceptables.

(a) ROC AUC. (b) PR AUC.

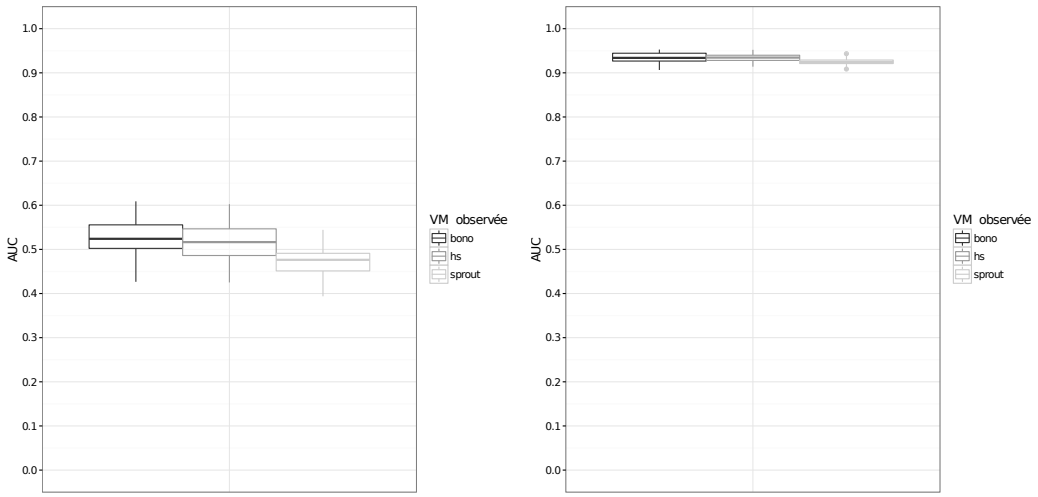


FIGURE 6.2.13 – Détection binaire de symptômes préliminaires à des violations de service avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*.

6.2.3 Détection de violations de service

La détection de violations de service renvoie des alarmes qu'un administrateur doit impérativement prendre en compte car le service est déjà dégradé du point de vue d'un utilisateur (voir chapitre 3).

Plusieurs tests ont été menés afin d'évaluer les performances de détection associées :

(a) ROC AUC. (b) PR AUC.

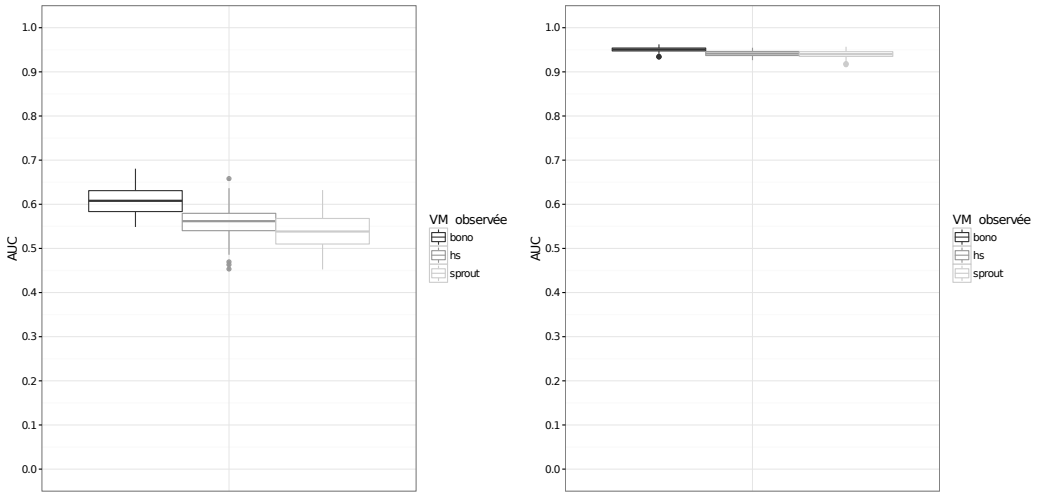


FIGURE 6.2.14 – Détection binaire de symptômes préliminaires à des violations de service avec un apprentissage sur $\mathcal{B}$ et une prédiction sur $\mathcal{D}$ (décalage de 1 semaine entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*.

— Nous présentons les résultats de détection avec le but de diagnostiquer la VM à l'origine d'une violation de service ainsi que dans le but de détecter le type d'erreur locale à l'origine de la violation.
— Une comparaison est réalisée sur les performances de détection avec un monitoring SE et hyperviseur. Elle permet d'évaluer si nos résultats sont meilleurs ou au moins aussi bons en utilisant la source de monitoring SE qui nécessite l'installation d'agents dans les VMs du service.
— Une étude est faite sur la sensibilité des performances au temps d'injection.
— Enfin, la validité de modèles pour une prédiction faite 1 mois et une prédiction 1 semaine après la collecte des données d'entrainement est présentée.

6.2.3.1 Détection avec diagnostic du type d'erreur à l'origine de la violation de service

Les figures 6.2.15 (a) et (b) représentent les ROC et PR AUC pour la détection de violations de service selon le type d'erreur locale à la VM observée qui est à l'origine de la violation de service.

Les erreurs sont détectées parfaitement dans toutes les VMs avec des valeurs ROC AUC en moyenne supérieures à 0.98. Les performances de détection sont meilleures que lorsque la même analyse est menée pour la détection d'erreurs avec diagnostic du type d'erreur(voir figure 6.2.2). Les PR AUC sont excellentes également avec des moyennes supérieures à 0.90.

(a) ROC AUC. (b) PR AUC.

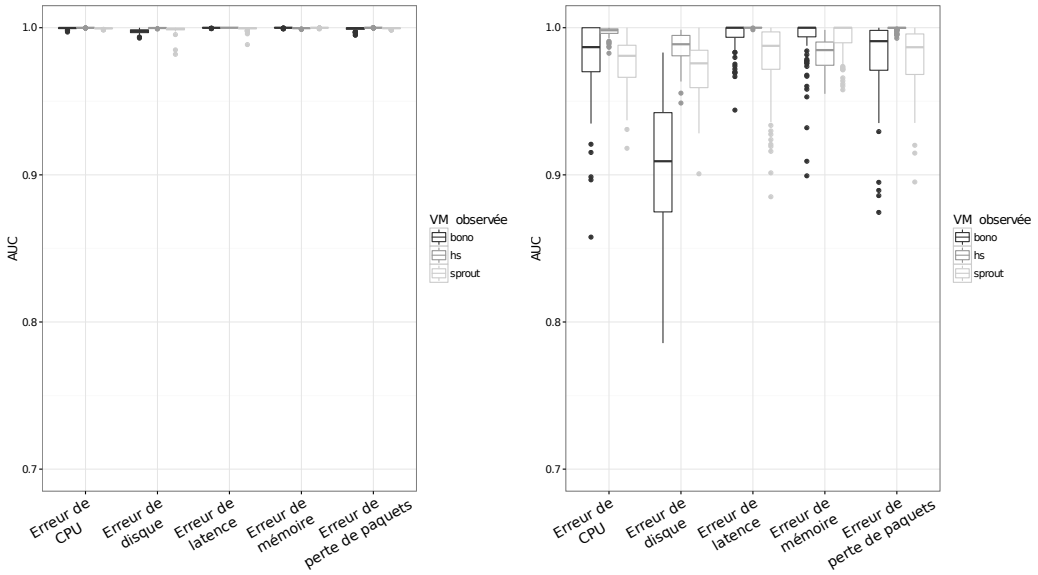


FIGURE 6.2.15 – Détection de violations de service selon le type d'erreur locale à la VM observée en *Detect_Sup_RF*.

6.2.3.2 Détection avec diagnostic de la VM à l'origine d'une violation de service

Les figures 6.2.16 représentent les ROC et PR AUC pour la détection de violations de service selon la VM à l'origine de la violation. Ces figures montrent qu'à partir des trois VMs il est à chaque fois possible de localiser l'origine d'une violation de service avec de très bonnes performances. Depuis chacune des VMs, les ROC AUC sont excellentes avec une moyenne supérieure à 0.98. Les PR AUC sont acceptables mais variables selon les VMs. Les meilleures performances sont obtenues pour Homestead qui à des PR AUC moyennes excellentes supérieures à 0.87.

Ainsi, les performances pour ce type de détection appliqué aux violations de service sont bons. Ils sont toutefois inférieurs à ceux obtenus pour ce même type de détection mais appliqué aux erreurs (voir partie 6.2.1.3). La conclusion est la même que dans la partie 6.2.1.3, à savoir que la VM à la fin de la chaine CSCF est la plus à même de prédire dans quelle VM se situe une anomalie en cours.

6.2.3.3 Comparaison avec le monitoring SE

Dans cette partie nous comparons les résultats de détection de violations de service selon la VM à l'origine de la violation, obtenus avec un monitoring SE. Les figures 6.2.17 (a) et (b) présentent les ROC et PR AUC calculées pour la détection de violations de service selon la VM à l'origine de la violation, en monitoring SE.

(a) ROC AUC. (b) PR AUC.

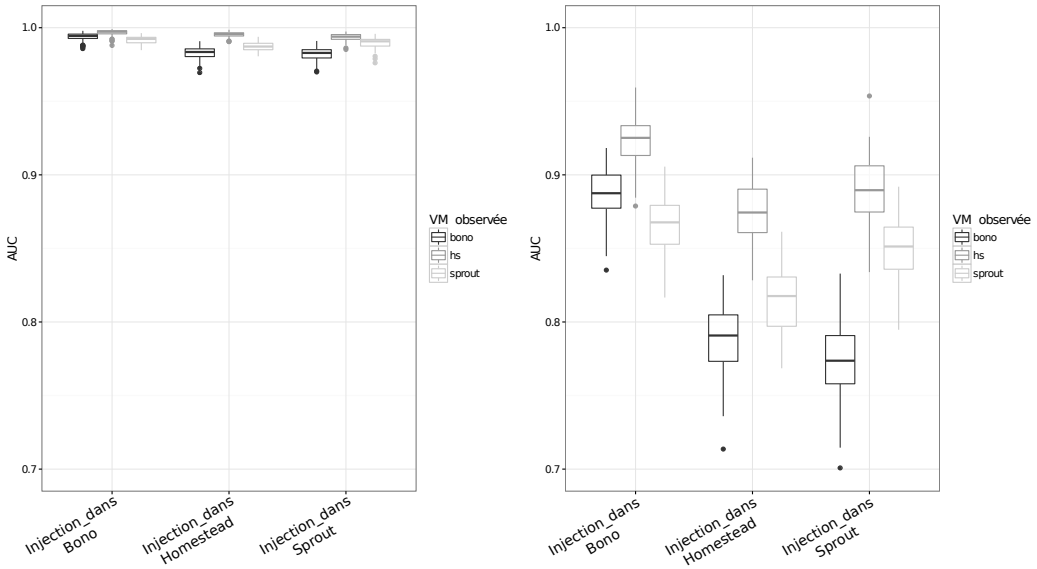


FIGURE 6.2.16 – Détection de violations de service selon la VM à l'origine de la violation en *Detect_Sup_RF*.

Ces résultats sont à comparer avec les figures 6.2.16 (a) et (b) qui correspondent à la même analyse mais avec les données du monitoring hyperviseur. Nous remarquons que les performances de détection sont supérieures avec le monitoring SE.

Les compteurs de performance supplémentaires qu'il est possible de collecter grâce au monitoring SE par rapport au monitoring hyperviseur apportent une amélioration significative des performances de détection par apprentissage supervisé, pour la détection des erreurs de notre modèle d'erreur.

6.2.3.4 Sensibilité à la durée d'injection

Les figures 6.2.18 (a) et (b) présentent les ROC et PR AUC calculées pour la détection de violations de service selon le type d'erreur locale à la VM observée pour des injections durant 4 min (jeu de données $\mathcal{C}$). Ces figures sont à comparer avec les figures 6.2.16 (a) et (b) qui présentent le même type d'analyse mais pour des injections de 10 min. Les performances de détection sont similaires entre des injections de 4 min et 10 min (dans le cas de la détection d'erreur, les performances étaient meilleures avec des injections de 10 min, voir les figures 6.2.6).

Nous pouvons donc conclure que lors d'une évaluation pour la détection de violations de service avec un entrainement et une prédiction sur un même jeu de données, la durée d'injection n'influence pas les résultats de détection de violations de service.

(a) ROC AUC. (b) PR AUC.

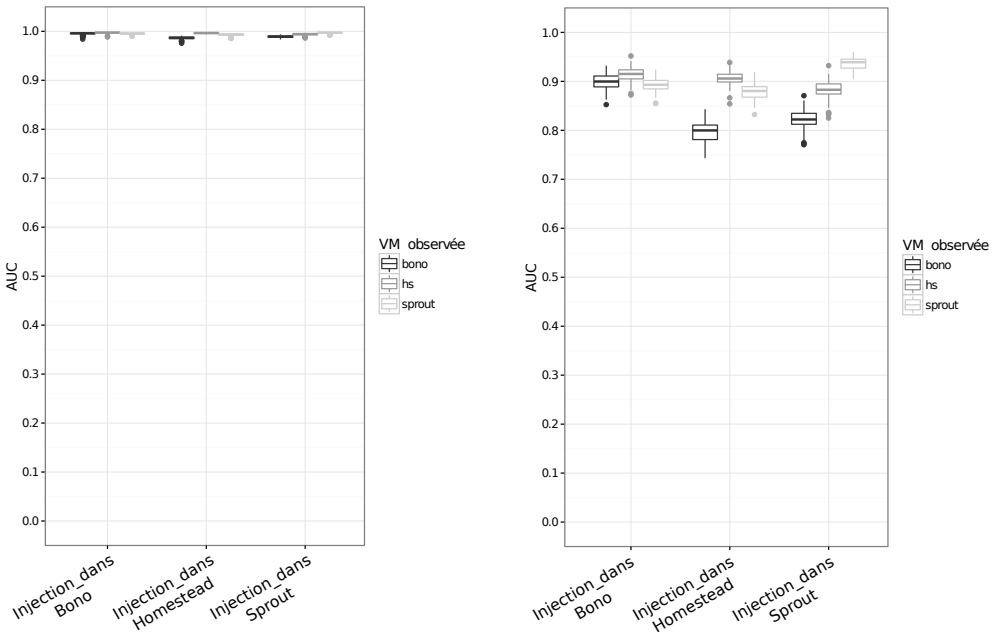


FIGURE 6.2.17 – Détection de violations de service selon la VM à l'origine de la violation en monitoring SE en *Detect_Sup_RF*.

6.2.3.5 Durée de validité du jeu d'entrainement

Les figures 6.2.19 (a) et (b) présentent les ROC et PR AUC calculées pour la détection de violations de service selon la VM à l'origine de la violation de service avec un entrainement sur le jeu de données $\mathcal{A}$ et une prédiction sur le jeu de données $\mathcal{B}$. La prédiction a donc lieu sur des données collectées 1 mois après la collecte des données d'entrainement des modèles. Ces figures sont à comparer avec les figures 6.2.16 (a) et (b) qui présentent le même type d'analyse mais pour une prédiction sur $\mathcal{A}$.

Les figures montrent d'excellentes performances de détection vis-à-vis des ROC AUC moyennes. La détection d'un mois à l'autre conserve un bas taux de faux positifs car les ROC AUC demeurent en moyenne supérieures à 0.80. Toutefois, il y a une grande perte de précision et de rappel par rapport au cas 6.2.16 qui n'est pas acceptable pour notre contexte d'étude.

Les résultats sont similaires pour une détection de violations de service selon le type d'erreur locale à la VM observée et pour une détection binaire pour un entrainement et une prédiction sur ces mêmes jeux de données. Les résultats dans le cas d'une prédiction a lieu sur des données collectées 1 mois après la collecte des données d'entrainement des modèles sont acceptables avec des ROC AUC moyennes d'environ 0.70, toutefois les écarts types des AUC sont importants. Ces résultats ne sont pas présentés.

(a) ROC AUC. (b) PR AUC.

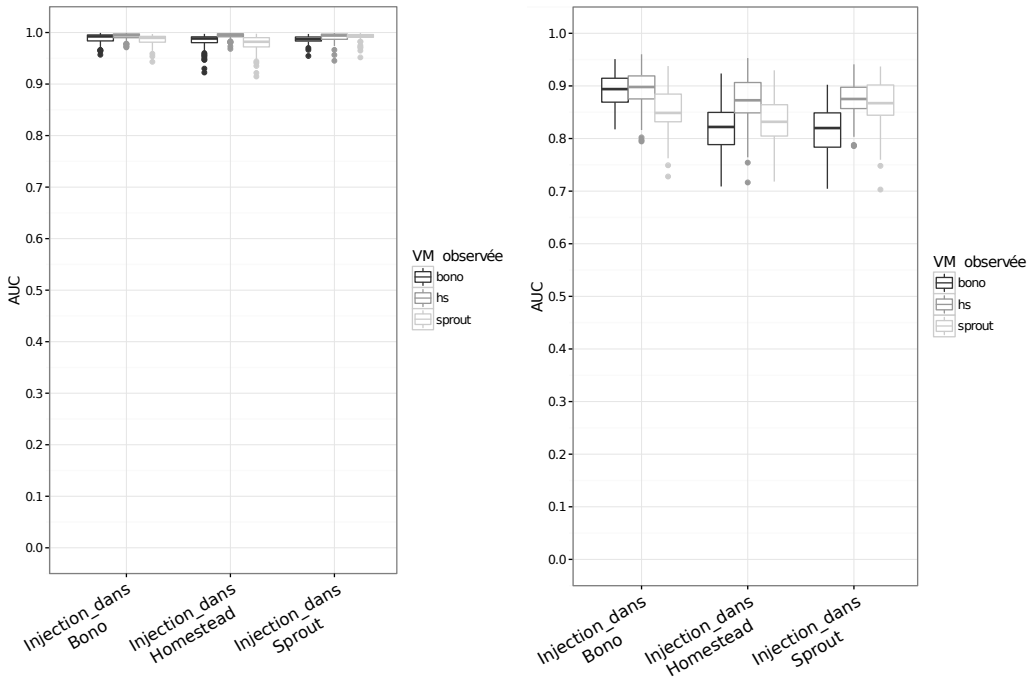


FIGURE 6.2.18 – Détection de violations de service selon le type d'erreur locale à la VM observée et 4 min d'injection en *Detect_Sup_RF*.

De cela nous concluons de nouveau que nos modèles ne sont pas adaptés pour la détection en ligne de violation de service sans actualisation régulière des modèles de détection.

Nous faisons de plus la comparaison avec les résultats obtenus à partir des jeux de données en monitoring SE. Les figures 6.2.20 (a) et (b) présentent les ROC et PR AUC calculées pour la détection de violations de service selon le type d'erreur locale à la VM observée avec un entrainement sur le jeu de données $\mathcal{A}$ et une prédiction sur le jeu de données $\mathcal{B}$. La prédiction a donc lieu sur des données collectées 1 mois après la collecte des données d'entrainement des modèles.

Les performances de prédiction ne sont pas acceptables vis-à-vis des PR AUC moyennes en monitoring SE. Les performances en détection binaire associées ne sont quant à elles pas acceptables vis-à-vis des ROC AUC moyennes.

6.3 Évaluation de la détection DESC hybride

Cette section présente les tests menés pour la détection d'anomalies par détection DESC hybride appliquée avec l'algorithme Random Forest (*Detect_DESC_hybride*). Plus précisément, l'analyse se fait par détection DESC hybride agrégée (voir l'explication en 3.3.4). Ces tests sont réalisés sur deux niveaux

(a) ROC AUC. (b) PR AUC.

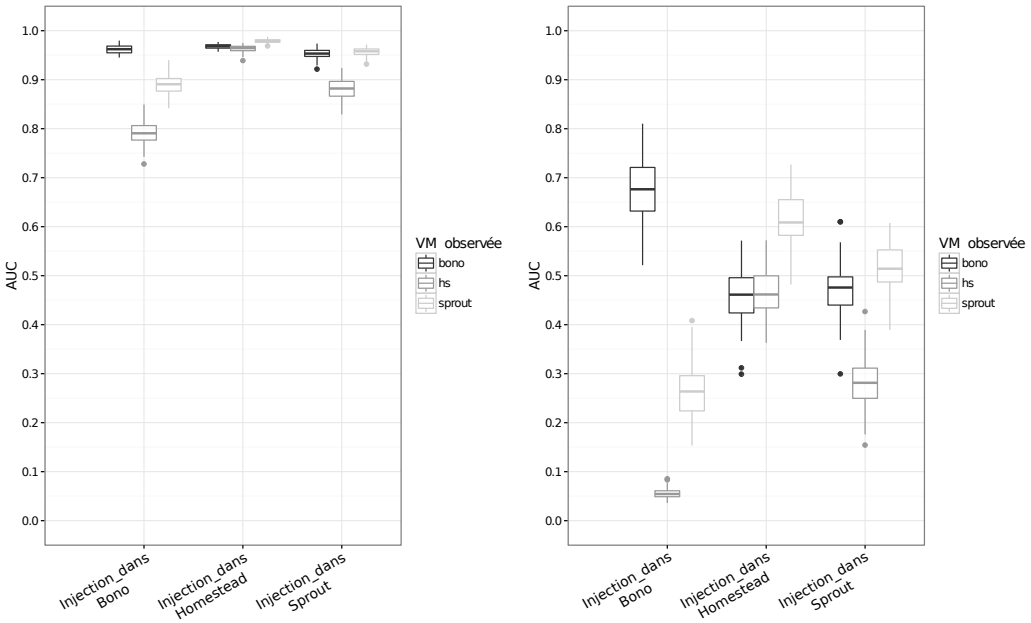


FIGURE 6.2.19 – Détection de violations de service selon la VM à l'origine de la violation de service avec un apprentissage sur $\mathcal{A}$ et une détection sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*.

de détection qui sont les erreurs et les violations de service, ainsi que sur un type de détection qu'est la détection selon l'erreur locale à la VM observée.

Nous signalons que les tests concernant le type de détection selon la VM à l'origine d'une anomalie ont été menés mais ne donnent pas de performances de détection acceptables. De même les tests concernant la détection de symptômes préliminaires à une violation de service ne donnent pas de performances de détection acceptables. Ces tests ne sont donc pas présentés.

6.3.1 Détection d'erreurs

Les figures 6.3.1 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon le type d'erreur locale à la VM observée. Les résultats sont excellents si l'on observe les ROC AUC moyennes pour toutes les erreurs détectées. En terme de PR AUC, les erreurs de latence réseau et de perte de paquets possèdent les moins bonnes performances. Une faible PR AUC moyenne pour ces deux erreurs avait également été constatée dans le cas d'un entrainement partiel sur peu d'exemples d'intensités d'injection (voir les figures 6.2.4) et dans le cas de détection d'erreurs en détection par apprentissage supervisé selon le type d'erreur (voir les figures 6.2.9).

(a) ROC AUC. (b) PR AUC.

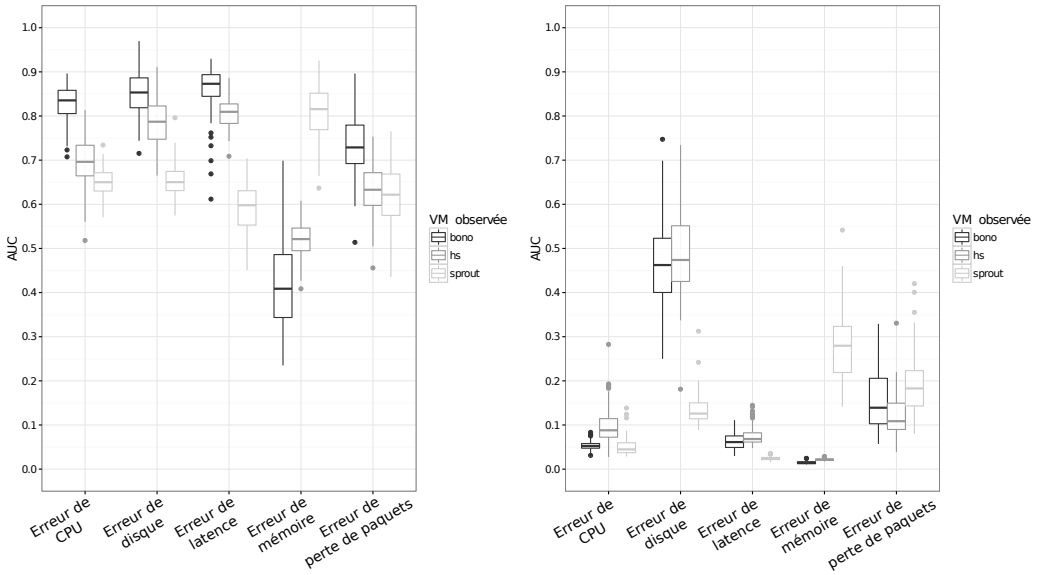


FIGURE 6.2.20 – Détection de violations de service selon le type d'erreur locale à la VM observée en monitoring SE avec un apprentissage sur $\mathcal{A}$ et une détection sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_Sup_RF*.

À titre comparatif, les figures 6.3.2 (a) et (b) présentent les ROC et PR AUC calculées pour la détection d'erreurs selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{B}$.

Dans le cas de détection sur un jeu de données avec moins d'intensités d'injection à détecter, les performances en terme de PR AUC moyenne sont acceptables voire excellentes à part pour la détection des erreurs de mémoire par les VMs Sprout et Homestead.

Cela nous indique que la configuration de notre algorithme de clustering tel qu'il est dans notre implémentation ne permet pas de détecter les erreurs de basses intensités d'injection (i.e. de 1 à 3). Dans notre contexte, ce point n'est pas pénalisant car les erreurs de basses intensités d'injection ont moins de chance d'entrainer des symptômes préliminaires de violation de service que les injections de fortes intensités. Les injections de forte intensité sont en effet des erreurs qui peuvent amener plus rapidement un système à manquer de ressource et à être sujet à une violation de service comme mentionné dans la sous section 4.4.2 du chapitre 4. Afin de mieux les détecter, il est nécessaire d'augmenter la valeur du facteur de vieillissement *lambda*.

(a) ROC AUC. (b) PR AUC.

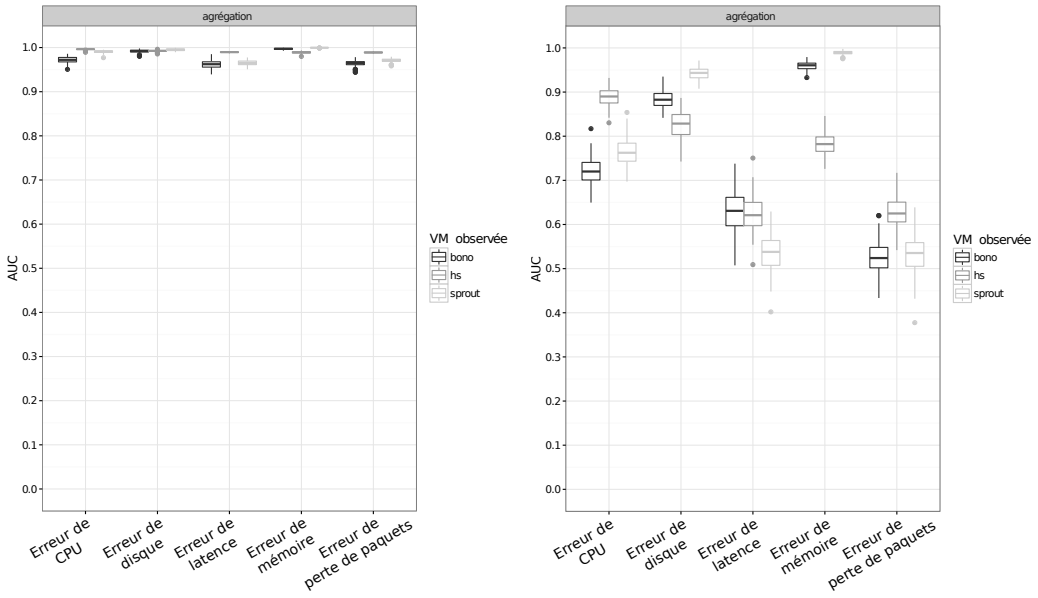


FIGURE 6.3.1 – Détection d'erreurs selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{A}$ en *Detect_DESC_hybride*.

6.3.2 Détection de violations de service

Notre détection DESC hybride permet également de détecter des violations de service selon le type d'erreur locale à la VM observée. Les figures 6.3.3 (a) et (b) en montre les performances de détection. Les PR AUC calculées sont très variables. Leur valeur moyenne pour chaque erreur reste toutefois acceptable pour les VMs Sprout et Homestead mais pas pour Bono pour les erreurs de disque, de latence et de perte de paquets. Les ROC AUC moyennes sont quant à elles excellentes car elles sont supérieures à 0.95.

La détection de violations de service selon le type d'erreur locale est donc envisageable par détection DESC hybride car elle amène peu de faux positifs, toutefois de nombreux vrais positifs sont manqués surtout dans la VM Bono.

6.3.3 Durée de validité du jeu d'entrainement

Les figures 6.3.4 (a) et (b) présentent les ROC et PR AUC calculées pour la détection binaire d'erreurs locale à la VM observée avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$. La prédiction a donc lieu sur des données collectées 1 mois après la collecte des données d'entrainement du modèle.

Les performances sont excellentes en terme de PR AUC avec une valeur moyenne supérieure à 0.90. Elles sont acceptables mais toutefois un peu faibles en terme de ROC AUC car entre 0.70 et 0.74, ce qui veut dire que notre méthode amène de

(a) ROC AUC. (b) PR AUC.

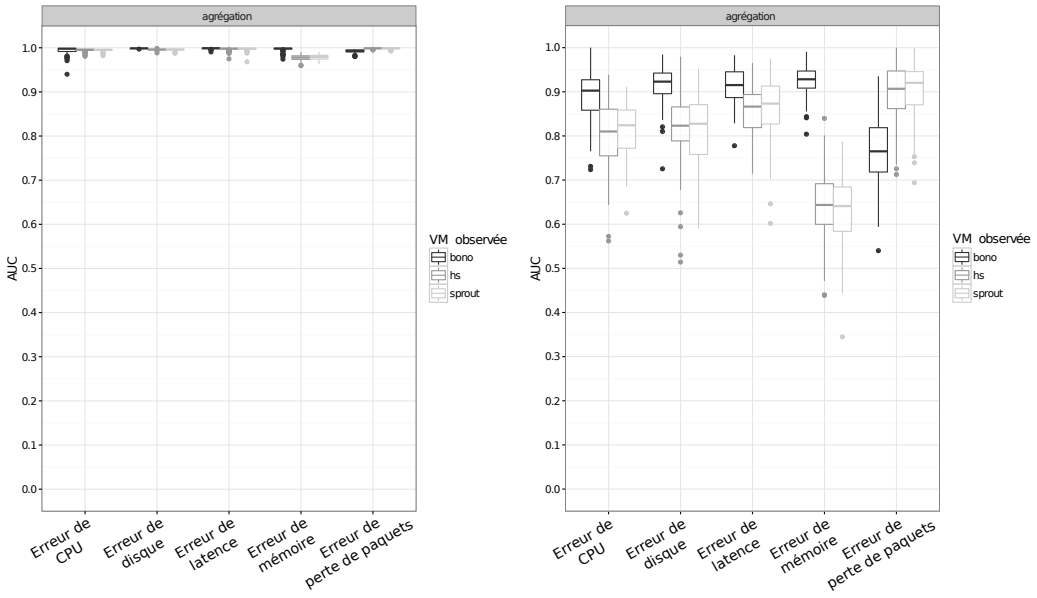


FIGURE 6.3.2 – Détection d'erreurs selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{B}$ en *Detect_DESC_hybride*.

(a) ROC AUC. (b) PR AUC.

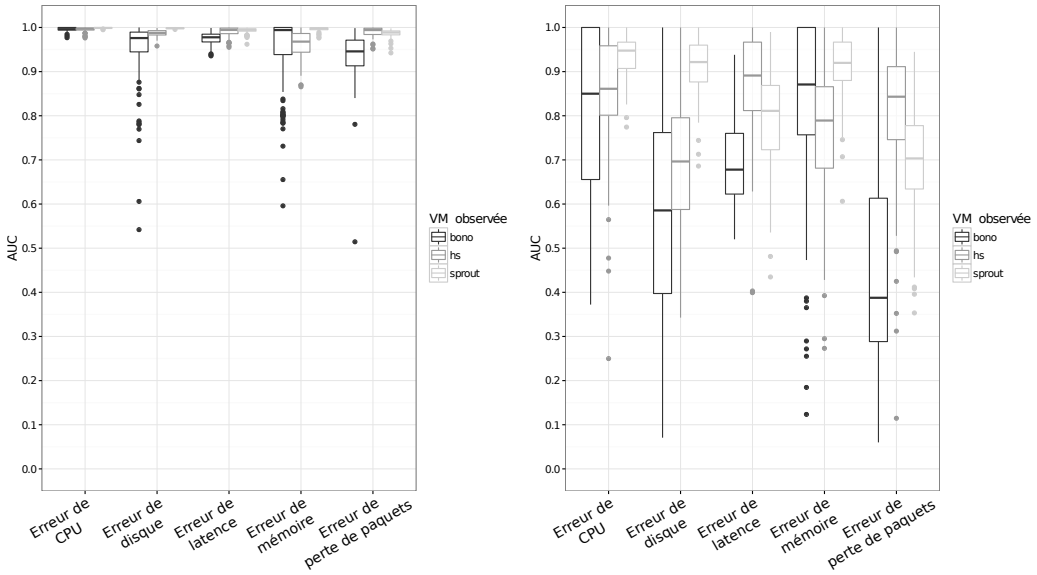


FIGURE 6.3.3 – Détection de violations de service selon le type d'erreur locale à la VM observée sur le jeu de données $\mathcal{B}$ en *Detect_DESC_hybride*.

(a) ROC AUC. (b) PR AUC.

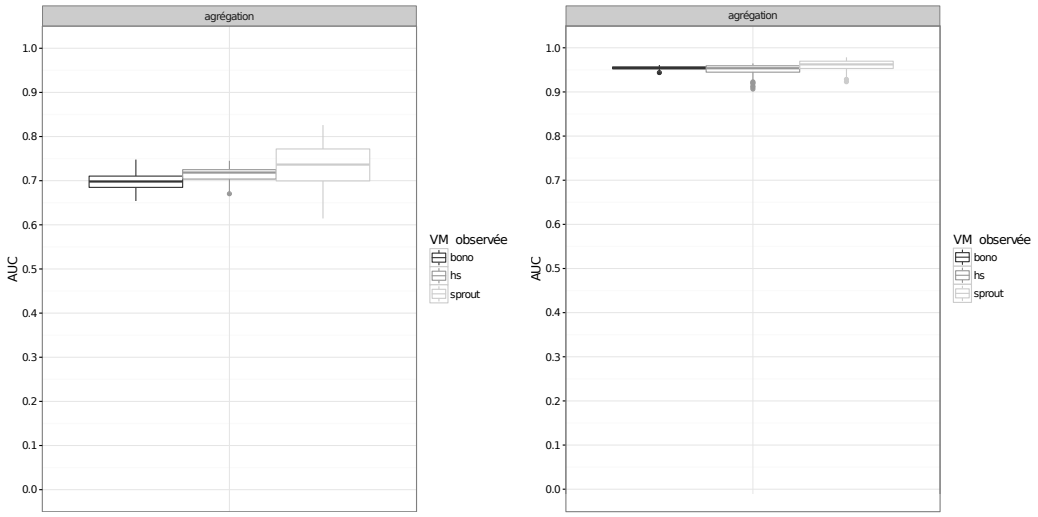


FIGURE 6.3.4 – Détection binaire d'erreurs avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_DESC_hybride*.

nombreux faux positifs. Les performances obtenues sont de plus inférieures à celles de la détection par apprentissage supervisé (voir les figures 6.2.8).

Les résultats la détection binaire de violation de service avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ sont présentés dans les figures 6.3.5 (a) et (b). Dans ce cas, la détection n'est pas envisageable car les ROC AUC de Bono et Sprout ne sont pas acceptables. Des résultats similaires ont été obtenus pour le même test effectué avec une détection par apprentissage supervisé dans la section précédente.

Les performances de détection ne sont pas acceptables vis-à-vis des PR AUC pour des détections avec diagnostics d'erreur ou de violation de service et ne sont pas présentés.

La détection par détection DESC hybride avec une prédiction qui a lieu sur des données collectées 1 mois après la collecte des données d'entrainement de modèles est donc acceptable pour la détection binaire d'erreurs mais pas pour la détection de violations de service (à part dans la VM Bono).

6.4 Analyse des compteurs de performance utiles à la détection

Nous rapellons que nous cherchons ici à déterminer s'il est possible de généraliser l'usage de certains compteurs de performance dans la détection d'un certain type d'erreur ou dans la détection dans une VM en particulier. Cela reviendrait à établir

(a) ROC AUC. (b) PR AUC.

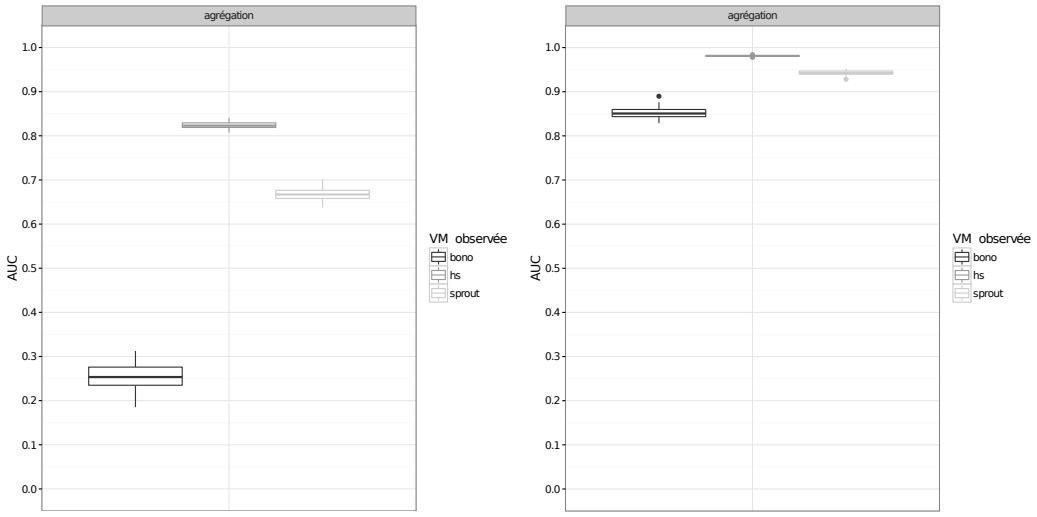


FIGURE 6.3.5 – Détection binaire de violation de service avec un apprentissage sur $\mathcal{A}$ et une prédiction sur $\mathcal{B}$ (décalage de 1 mois entre les dates des données d'entrainement et de prédiction) en *Detect_DESC_hybride*.

un "dictionnaire d'anomalies".

Nous avons collecté les 10 compteurs de performance les plus utilisés dans les arbres des modèles basés sur l'algorithme Random Forest pour nos trois types de détection. Le jeu de données $\mathcal{A}$ est utilisé pour obtenir des résultats suivants. Ces compteurs sont présentés dans six tableaux :

- le tableau 6.1 avec des données en monitoring hyperviseur pour la détection d'erreurs et de violation de service binaire,
- le tableau 6.2 avec des données en monitoring hyperviseur pour la détection d'erreurs et de violation de service selon la VM à l'origine (avec diagnostic),
- le tableau 6.3 avec des données en monitoring hyperviseur pour la détection d'erreurs et de violation de service selon le type d'erreur locale à la VM observée (avec diagnostic),
- le tableau 6.4 avec des données en monitoring hyperviseur pour les détections binaire et selon la VM à l'origine de l'erreur de symptômes préliminaires à une violation de service,
- le tableau 6.5 avec des données en monitoring hyperviseur pour la détection d'erreurs selon le type d'erreur locale à la VM observé et en séparant les compteurs utiles selon le type d'erreur qui est détecté par les compteurs.
- le tableau 6.6 avec des données en monitoring SE pour la détection d'erreurs selon le type d'erreur locale à la VM observé et en séparant les compteurs utiles selon le type d'erreur qui est détecté par les compteurs.

L'analyse des compteurs de performance grâce au code couleur présenté dans la

section 4.6 nous permet de constater plusieurs points qui sont les suivants :

- Les compteurs réseau sont majoritairement présents dans la détection binaire de violations de service (voir tableau 6.1).
- Les compteurs CPU sont majoritairement présents dans la détection binaire d'erreurs (voir tableau 6.1).
- Aucun compteur réseau n'apparait pour la détection binaire ou avec diagnostic d'erreur (voir tableaux 6.1 et 6.2).
- Les compteurs disque sont majoritairement présents dans la détection avec diagnostic, que ce soit d'erreur ou de violation de service (voir tableau 6.2).
- Les compteurs mémoire participent majoritairement à la détection d'erreurs ou de violations de service pour Homestead (voir tableaux 6.1 et 6.2) surtout avec les compteurs *Overhead_MBytes* (mémoire allouée à une VM au delà de la réservation mémoire de cette VM sur l'hyperviseur) et *mem_entitlement* (place en mémoire de l'hyperviseur réservée à la VM).
- Les compteurs d'énergie sont présents pour la détection binaire et avec diagnostic d'erreur pour Bono et Sprout (voir tableaux 6.1 et 6.2).
- Les détections selon le type d'erreur locale à la VM observée d'erreur ou de violation de service utilisent plusieurs types de compteurs (voir tableau 6.3). La détection dans Bono semble être efficace grâce aux compteurs CPU, et celle dans Homestead grâce aux compteurs disque.
- La détection de symptômes préliminaires de violation de service dans Bono semble être efficace grâce aux compteurs CPU (de manière similaire à la détection selon le type d'erreur locale à la VM observe, voir point précédent), et celle dans Sprout grâce aux compteurs réseau (voir tableau 6.4).
- Les compteurs mémoire *Overhead_MBytes* et *mem_llSwapOutRate* (taux auquel la mémoire est swappée de la mémoire active à la mémoire cache de l'hyperviseur) sont présents dans les deux détections binaire et avec diagnostic de symptômes préliminaires.
- Les compteurs liés au type d'une erreur aident majoritairement la détection et le diagnostic de ce type (voir tableau 6.5). Certains cas font toutefois exception. Par exemple les compteurs CPU sont majoritairement utiles à la détection des erreurs de disque dans Bono. Les compteurs mémoire n'apparaissent pas dans la liste des compteurs utiles à la détection des erreurs mémoire dans Homestead.
- Les compteurs CPU du monitoring SE aident beaucoup plus à la détection d'erreurs de CPU que les compteurs CPU du monitoring hyperviseur, toutes VMs confondues (voir tableau 6.6).
- Les compteurs mémoire du monitoring SE aident beaucoup plus à la détection d'erreurs de mémoire que les compteurs mémoire du monitoring hyperviseur, toute VMs confondues.
- Les compteurs réseau SE sont beaucoup plus présents pour la détection de perte de paquets que les compteurs réseau du monitoring hyperviseur, toutes VMs confondues. Les compteurs utiles réseau du monitoring SE décrivent de plus des caractéristiques de protocoles réseau beaucoup plus bas niveau

que les compteurs réseau du monitoring hyperviseur.

La comparaison du tableau des compteurs utiles avec un monitoring SE 6.6 avec l'analyse des compteurs utiles faite pour le cas d'étude MongoDB (voir sous section), montrent que certains compteurs sont commun à la détection des erreurs. En tout, Bono a 19 compteurs en commun avec les compteurs de la VM primaire MongoDB étudiée, Sprout en a 23 et Homestead en a 21 (sur 50 compteurs de performance présentés dans chaque tableau). Le type d'erreur qui est détecté avec un maximum de compteurs en commun entre les deux cas d'étude est l'erreur de CPU, avec entre 6 et 8 (atteint pour Sprout) compteurs en commun sur 10. L'erreur de perte de paquets est le type avec le moins de compteur en commun même si dans les deux cas les compteurs réseau sont présents majoritairement. Les tâches opérées par Bono, Homestead, Sprout et le VM primaire MongoDB sont à ce point différentes que les compteurs de performance utilisés pour la détection ne sont pas les mêmes.

Ainsi, les erreurs se manifestent davantage de par leur activité CPU différente si l'on considère une classification binaire. Cela est cohérent avec l'implémentation de nos injections d'erreur de CPU, mémoire et disque car chacun de ces types suppose respectivement un CPU actif travaillant pour accroitre la consommation CPU, un CPU actif travaillant à organiser de nombreuses allocations mémoire et un CPU inactif attendant l'accomplissement de nombreux accès disque. De plus, les pertes de paquets amènent également à une charge CPU plus importante car elle forcent le système à traiter plusieurs fois les mêmes envois de paquets alors que d'autres envois sont toujours plus nombreux à faire en parallèle.

Lorsque l'on veut dissocier les types d'erreur selon la VM à l'origine de l'erreur, des compteurs de performance variés de disque, mémoire, CPU et consommation d'énergie sont utilisés. Les compteurs liés au réseau ont une moindre importance pour la détection d'erreurs binaire ou avec diagnostic selon la VM à l'origine de l'erreur car ils n'apparaissent pas dans les tableaux correspondants (voir tableaux 6.1 et 6.2). Il sont utiles lors de la dissociation des erreurs détectées selon leur type (voir tableau 6.3)

Les violations de service se manifestent quant à elles par des échanges réseau différents à la fois si l'on considère une classification binaire et une classification avec diagnostic. Une variété plus importante de compteurs de performance est toutefois ici aussi nécessaire à la différenciation des classes de violation de service selon si la violation est due à une défaillance locale ou non.

Nous remarquons également que les compteurs de performance utiles à la détection sont dépendants des VMs étudiées et de leur rôle. Ceci est constaté pour les trois niveaux de détection dans ce cas d'étude. Cette conclusion est retrouvée en comparant les compteurs utiles à la détection d'erreurs dans ce cas d'étude avec ceux du cas d'étude MongoDB.

Enfin, les compteurs CPU, mémoire et réseau du monitoring SE sont beaucoup plus présents dans la détection des erreurs respectivement de CPU, mémoire et réseau que lors de l'utilisation du monitoring hyperviseur. Cela peut être dû au fait que ces compteurs provenant du monitoring SE sont plus précis que ceux obtenus par l'hyperviseur. Les compteurs réseau du monitoring SE sont quant à eux

VM observée	Détection binaire de violation	Détection binaire d'erreur
Bono	rescpu_actpk15 net_packetsTx_nic2 cpu_ready_1 net_multicastTx datastore_numberWriteAveraged mem_swapinRate net_bytesRx_nic2 net_received_nic2 net_bytesTx_nic1 net_packetsTx	cpu_wait cpu_idle datastore_maxTotalLatency power_energy mem_activewrite power_power cpu_demand cpu_latency virtualDisk_writeLatencyUS rescpu_runpk1
Sprout	rescpu_actpk15 cpu_wait disk_read net_multicastTx net_multicastTx_nic1 net_packetsRx_nic1 net_bytesRx_nic2 cpu_run_1 net_bytesTx_nic1 mem_zipped	cpu_run_1 cpu_idle cpu_demand rescpu_runav1 cpu_used_1 cpu_wait_1 cpu_idle_0 power_energy disk_maxTotalLatency cpu_latency
Hs	net_multicastTx_nic1 rescpu_runav15 rescpu_actpk5 net_transmitted_nic1 cpu_idle_1 cpu_run mem_overheadTouched Overhead_UW_MBytes virtualDisk_totalWriteLatency net_multicastTx	mem_overheadTouched mem_compressed cpu_maxlimited_1 disk_busResets disk_write cpu_run_1 cpu_overlap_1 datastore_numberReadAveraged6 disk_write mem_compressionRate

Tableau 6.1 – Compteurs les plus utiles en monitoring hyperviseur en détection binaire d'erreurs et de violation de service.

VM observée	Détection de la VM à l'origine d'une violation	Détection de la VM à l'origine d'une erreur
Bono	net_packetsTx virtualDisk_writeLatencyUS disk_maxTotalLatency net_packetsTx_nic1 cpu_system_0 net_bytesTx_nic1 datastore_maxTotalLatency mem_entitlement net_transmitted net_bytesTx	virtualDisk_writeLatencyUS mem_activewrite disk_maxTotalLatency cpu_idle power_energy virtualDisk_totalWriteLatency cpu_demand mem_entitlement rescpu_runav1 datastore_maxTotalLatency
Sprout	virtualDisk_writeLatencyUS cpu_idle virtualDisk_writeIOSize net_bytesRx_nic1 net_bytesRx virtualDisk_numberWriteAveraged virtualDisk_writeOIO mem_entitlement cpu_idle_0 virtualDisk_totalWriteLatency	power_energy cpu_run_1 virtualDisk_writeLatencyUS disk_maxTotalLatency rescpu_runav1 datastore_maxTotalLatency rescpu_runav5 cpu_latency cpu_demand power_power
Hs	cpu_run cpu_run_1 rescpu_samplePeriod mem_entitlement virtualDisk_totalReadLatency datastore_numberReadAveraged6 disk_busResets disk_write net_broadcastRx_nic1 mem_overheadTouched	mem_compressed mem_compressionRate mem_entitlement rescpu_runav5 datastore_numberReadAveraged6 disk_write disk_busResets cpu_run_1 cpu_system_0 mem_overheadTouched

Tableau 6.2 – Compteurs les plus utiles en monitoring hyperviseur en détection avec diagnostic selon la VM à l'origine d'erreur et de violation de service.

VM observée	Détection du type d'erreur locale à l'origine d'une violation	Détection du type d'erreur locale à l'origine d'une erreur
Bono	cpu_used mem_entitlement mem_overheadTouched rescpu_runav5 mem_activewrite cpu_run cpu_run_1 rescpu_runav1 cpu_demand net_transmitted_nic1	rescpu_runav5 mem_entitlement cpu_demand rescpu_actpk1 mem_activewrite cpu_run mem_overheadTouched rescpu_runav1 rescpu_actpk15 net_transmitted
Sprout	rescpu_runav5 mem_entitlement mem_activewrite virtualDisk_largeSeeks_scsi0 virtualDisk_writeIOSize_scsi0 virtualDisk_numberWriteAveraged_scsi0 net_packetsRx_nic1 cpu_run_1 cpu_system_0 net_broadcastRx_nic1	mem_entitlement cpu_system_0 mem_activewrite rescpu_runav5 cpu_system net_packetsRx_nic1 virtualDisk_largeSeeks_scsi0 net_packetsRx disk_write cpu_run_1
Hs	net_multicastRx_nic1 disk_read_naa rescpu_actpk1 mem_llSwapOutRate net_bytesRx_nic2 disk_write virtualDisk_totalReadLatency_scsi0 cpu_run virtualDisk_smallSeeks_scsi0 virtualDisk_readLoadMetric_scsi0	net_multicastRx_nic1 disk_write disk_read_naa power_power rescpu_actpk1 net_transmitted_nic1 mem_llSwapOutRate virtualDisk_totalReadLatency_scsi0 datastore_numberReadAveraged cpu_run

Tableau 6.3 – Compteurs les plus utiles en monitoring hyperviseur en détection multi-classe selon le type d'erreur locale à la VM observée et de violation de service.

	Détection de la VM source de symptômes préliminaires	Détection binaire de symptômes préliminaires
Bono	mem_overheadTouched cpu_used mem_entitlement rescpu_runav15 rescpu_actav5 rescpu_actpk15 mem_activewrite net_packetsRx rescpu_actpk5 rescpu_runpk15	rescpu_runav15 mem_activewrite net_bytesTx_nic1 rescpu_actpk15 mem_entitlement mem_overheadTouched net_packetsRx_nic1 rescpu_actpk5 cpu_used rescpu_actav5
Sprout	mem_entitlement net_bytesRx_nic1 net_packetsRx cpu_run mem_overheadTouched cpu_used virtualDisk_writeIOSize_scsi0 net_packetsRx_nic1 net_packetsTx_nic1 net_received	net_received_nice1 cpu_used_1 net_bytesRx_nic1 virtualDisk_writeIOSize_scsi0 net_received mem_overheadTouched cpu_demand mem_entitlement net_packetsRx_nic1 net_packetsTx_nic1
Hs	net_multicastRx_nic1 mem_llSwapOutRate datastore_write virtualDisk_numberReadAveraged_scsi0 cpu_entitlement cpu_run mem_compressed Overhead_MBytes rescpu_samplePeriod datastore_numberReadAveraged	net_multicastRx_nic1 virtualDisk_numberReadAveraged_scsi0 Overhead_MBytes datastore_write mem_llSwapOutRate datastore_numberReadAveraged rescpu_samplePeriod cpu_entitlement mem_compressed cpu_run

Tableau 6.4 – Compteurs les plus utiles en monitoring hyperviseur en détection binaire et multi-classe de symptômes préliminaires de violations de service.

Classe d'erreur	Détection du type d'erreur locale à l'origine d'une erreur avec des données de monitoring hyperviseur		
	Bono	**Sprout**	**Homestead**
Perte de paquets	net_packetsTx	net_received_nic1	net_multicastRx_nic1
	rescpu_runpk1	net_bytesRx_nic1	disk_read_naa
	net_bytesRx_nic1	net_received	net_bytesRx_nic2
	net_transmitted_nic1	net_bytesRx	disk_commands_naa
	mem_overheadTouched	net_broadcastRx_nic1	mem_overheadTouched
	net_packetsTx_nic1	disk_write	net_broadcastRx_nic1
	mem_entitlement	net_packetsRx	mem_llSwapOutRate
	net_bytesTx_nic1	net_packetsRx_nic1	virtualDisk_writeLoadMetric_scsi0
	rescpu_actpk5	virtualDisk_largeSeeks_scsi0	datastore_read
	cpu_used	mem_entitlement	cpu_run
Latence	cpu_used	net_packetsTx	net_multicastRx_nic1
	net_bytesRx_nic1	cpu_system_1	rescpu_actpk1
	net_packetsRx	cpu_system_0	disk_read_naa
	net_transmitted_nic1	cpu_system	disk_read
	net_bytesTx_nic1	net_packetsTx_nic1	net_transmitted_nic1
	net_bytesRx	mem_entitlement	mem_overheadTouched
	cpu_demand	disk_write	datastore_writeg
	net_transmitted	net_packetsRx	mem_llSwapOutRate
	net_bytesTx	net_packetsRx_nic1	rescpu_samplePeriod
	net_received	mem_activewrite	cpu_run
Mémoire	rescpu_runpk15	rescpu_runav15	virtualDisk_smallSeeks_scsi
	mem_activewrite	mem_activewrite	cpu_costop_1
	mem_overheadTouched	rescpu_runav5	net_multicastRx_nic1
	mem_entitlement	mem_overheadTouched	rescpu_runpk15
	rescpu_actpk15	mem_entitlement	cpu_wait_1
	rescpu_actav15	rescpu_actpk15	net_broadcastRx_nic1
	disk_write	rescpu_actav15	rescpu_actpk15
	virtualDisk_write_scsi	virtualDisk_readOIO_scsi	net_droppedTx
	virtualDisk_mediumSeeks_scsi	virtualDisk_mediumSeeks_scsi0	datastore_totalWriteLatency
	cpu_used	disk_write	rescpu_actav5
Disque	cpu_wait	virtualDisk_numberWriteAveraged_scsi0	net_packetsTx
	rescpu_runpk1	cpu_system_0	rescpu_runpk1
	cpu_used_0	virtualDisk_writeIOSize_scsi0	cpu_ready_1
	rescpu_actpk1	virtualDisk_writeLoadMetric_scsi0	disk_write_naa
	cpu_used	virtualDisk_writeOIO_scsi0	power_power
	cpu_idle	disk_write	cpu_run_1
	cpu_demand	virtualDisk_write_scsi0	virtualDisk_totalReadLatency_scsi
	rescpu_runav1	virtualDisk_mediumSeeks_scsi0	datastore_numberReadAveraged
	rescpu_actpk5	disk_maxTotalLatency	mem_entitlement
	cpu_run	virtualDisk_totalWriteLatency_scsi0	net_packetsRx

Classe d'erreur	Détection du type d'erreur locale à l'origine d'une erreur		
	Bono	**Sprout**	**Homestead**
CPU	cpu_wait	cpu_run	cpu_maxlimited_1
	mem_activewrite	mem_activewrite	power_power
	cpu_idle_1	cpu_run_1	net_multicastRx_nic1
	cpu_demand	cpu_run_0	rescpu_actpk1
	net_packetsTx_nic1	cpu_demand	disk_read_naa
	cpu_wait_0	mem_entitlement	virtualDisk_readLoadMetric_scsi0
	rescpu_runav1	rescpu_runav1	cpu_system_0
	cpu_run0	virtualDisk_write_scsi	net_transmitted_nic1
	cpu_latency	cpu_used_1	datastore_numberReadAveraged
	cpu_run	cpu_run	net_multicastRx

Tableau 6.5 – Compteurs les plus utiles en monitoring hyperviseur en détection multi-classe selon le type d'erreur locale à la VM observée

beaucoup plus nombreux. Ces deux points amènent le monitoring SE à fournir de meilleures performances de détection que celles obtenues avec le monitoring hyperviseur. Il serait ainsi intéressant d'identifier quels compteurs de performances provenant du monitoring SE seraient succeptibles d'améliorer les performances de détection avec monitoring hyperviseur s'ils étaient ajoutés aux observations de monitoring hyperviseur.

6.5 Conclusion

Dans ce dernier chapitre nous avons évalué notre stratégie de détection sur le cas d'étude Clearwater et plus particulièrement sa chaîne CSCF composée d'une VM proxy, une VM routeur et une VM BDD Cassandra. Nous constatons qu'une détection simple par apprentissage supervisé peut être très efficace dans notre contexte de détection. D'excellentes performances sont obtenues en utilisant l'algorithme Random Forest. Cet algorithme a l'avantage de ne demander quasiment pas d'étude préalable des données.

Les différents tests menés nous ont permis de faire des constats quant aux limites des performances de détection de cette stratégie. L'exécution de ces tests est nécessaire à toute nouvelle étude afin de s'assurer du contexte des performances de détection de modèles d'apprentissage automatique.

Concernant l'approche par apprentissage supervisé, nos trois niveaux de détection d'anomalies (erreur, symptômes préliminaires à une violation de service, et violation de service) ont été évalués et se sont avérés donner globalement de bons résultats chacun pour au moins un type de détection (i.e., détection binaire et détection permettant le diagnostic du type d'erreur locale et de la VM à l'origine de l'anomalie). Il s'avère de plus que la détection à partir des données de monitoring SE donnent de meilleures performances de détection que celles constatées avec le monitoring hyperviseur.

Le frein à l'utilisation de cette technique est la durée de validité des modèles de

Classe d'erreur	Détection du type d'erreur locale à l'origine d'une erreur avec des données de monitoring SE		
	Bono	**Sprout**	**Homestead**
Perte de paquets	tcpext__tcprenorecoveryfail	tcp__retranssegs	tcp__retranssegs
	tcpext__tcpdsackignorednoundo	tcpext__tcprenorecoveryfail	tcpext__tcprenorecoveryfail
	tcpext__tcpabortontimeout	tcpext__tcpslowstartretrans	tcpext__tcpdsackignorednoundo
	tcp__outrsts	tcpext__tcpdsackignorednoundo	load__fifteen
	ip__indelivers	tcpext__tcpabortondata	proc__total
	tcpext__tcplossundo	tcpext__tcpsackshiftfallback	tcpext__tcptimeouts
	tcpext__tcpdsackoforecv	tcpext__tcptimeouts	tcpext__tcplossundo
	tcp__retranssegs	tcpext__tcplossundo	tcpext__tcpdsackoforecv
	tcp__retrans__percentage	tcpext__tcpdsackoforecv	cpu__system
	mem__mapped	mem__cached	tcp__retrans__percentage
Latence	tcpext__tcphpacks	mem__cached	rx__pkts__lo
	bytes__out	disk__free__absolute__rootfs	disk__free__absolute__rootfs
	mem__buffers	tcpext__delayedacks	tcp__activeopens
	tx__bytes__lo	load__fifteen	part__max__used
	tcp__outrsts	proc__total	disk__free
	rx__pkts__lo	tcp__outrsts	tcp__attemptfails
	proc__total	mem__free	tcpext__embryonicrsts
	rx__pkts__eth0	mem__buffers	mem__cached
	tx__pkts__eth0	mem__mapped	mem__mapped
	tcp__activeopens	diskstat__sda__write__bytes__per__sec	disk__free__percent__rootfs
Mémoire	mem__cached	part__max__used	mem__free
	load__fifteen	disk__free__absolute__rootfs	mem__cached
	mem__free	mem__free	tcpext__tcpabortondata
	proc__total	proc__total	vm__pgmajfault
	cpu__sintr	swap__free	diskstat__sda__reads
	cpu__system	disk__free	cpu__system
	load__five	cpu__system	mem__buffers
	mem__mapped	mem__buffers	mem__mapped
	mem__buffers	mem__mapped	diskstat__sda__reads__merged
	mem__mapped	diskstat__sda__reads__merged	vm__pgpgin
Disque	entropy__avail	io__nwrite	disk__free__absolute__rootfs
	load__one	diskstat__sda__weighted__io__time	load__one
	load__fifteen	cpu__system	load__fifteen
	proc__total	io__writes	proc__total
	cpu__idle	diskstat__sda__io__time	io__nwrite
	cpu__system	mem__dirty	diskstat__sda__io__time
	io__writes	diskstat__sda__write__time	load__five
	load__five	diskstat__sda__write__bytes__per__sec	mem__dirty
	diskstat__sda__writes__merged	diskstat__sda__percent__io__time	diskstat__sda__writes__merged
	diskstat__sda__write__bytes__per__sec	proc__run	disk__free__percent__rootfs

Classe d'erreur	Détection du type d'erreur locale à l'origine d'une erreur avec des données de monitoring SE		
	Bono	**Sprout**	**Homestead**
CPU	mem_cached	load_one	cpu_system
	load_one	proc_total	load_one
	load_fifteen	cpu_idle	cpu_idle
	cpu_idle	cpu_user	cpu_user
	cpu_user	load_five	disk_free
	load_five	load_fifteen	proc_total
	cpu_system	mem_buffers	load_five
	mem_buffers	mem_mapped	mem_dirty
	mem_mapped	diskstat_sda_writes_merged	part_max_used
	pkts_out	disk_free_percent_rootfs	disk_free_percent_rootfs

Tableau 6.6 – Compteurs les plus utiles en monitoring SE et détection multi-classe selon le type d'erreur locale à la VM observée

détection. Dans nos cas d'étude, c'est la détection avec diagnostic de l'origine d'une anomalie (i.e. détection multi-classe) dont la durée de validité est la plus faible. Nous avons de plus constaté que la détection de violations de service donne de meilleures performances que la détection d'erreurs lorsque les prédictions ont lieu peu de temps après l'entrainement. Toutefois, les modèles de détection d'erreurs sont ceux qui donnent de meilleurs résultats après 1 mois d'utilisation. Un nouvel apprentissage suppose soit de lancer une nouvelle expérimentation avec la nouvelle configuration du système cible (cela suppose une duplication de la plateforme de services cloud) soit de stocker en continu les nouvelles observations du système et de noter manuellement (ou de manière pseudo-automatisée) toutes nouvelles erreurs détectées et toutes nouvelles erreurs manquées par le système de détection. Ainsi le nouveau jeu d'apprentissage du système de détection se forme au fur et à mesure de l'évolution du système. Le premier cas est lourd en maintenance et en terme de coût car il suppose une duplication de la plateforme de services cloud. Le second cas suppose l'existence d'un second système de collecte de confirmation d'erreur. Ce second cas est le plus adapté dans notre contexte car nous cherchons à minimiser les coûts, la maintenance ainsi que les actions humaines nécessaires à la détection.

Notre détection DESC hybride est aussi directe à mettre en place car elle possède une configuration simple. Les résultats sont moins bons qu'avec une détection par apprentissage supervisé car elle permet un moins bon diagnostic d'erreur et de violation de service (elle ne traite par les symptômes préliminaires à une violation de service). Ils sont tout de même acceptables notamment pour une détection binaire agrégée avec une prédiction faite sur des données collectées 1 mois après la collecte des données d'entrainement. En effet les performances en terme de précision et de rappel sont excellents même dans le cas d'une prédiction sur des données collectées 1 mois après la collecte des données d'entrainement.

Nous montrons également que les compteurs de performance utiles à la détection d'anomalies sont dépendants du type d'erreur à détecter mais également du niveau de détection et du type de détection. Ils sont de plus dépendants de la VM dont

les observations servent à la détection. Selon le rôle et les entités installées sur une VM, les compteurs de performance utiles à la détection d'anomalies peuvent être très différents. Il ne serait donc pas envisageable de considérer un service cloud distribué sur plusieurs VMs comme un tout possédant une activité unique qu'il est possible de modéliser grâce à l'apprentissage automatique sur des compteurs de performance.

Chapitre 7

Conclusions et perspectives

Bilan

Un des défis des fournisseurs de services cloud est d'assurer la disponibilité et la fiabilité de services à la fois différents, en allocation dynamique (service à la demande) et pour des utilisateurs aux demandes hétérogènes. Ce défi est d'autant plus important que les utilisateurs demandent à ce que les services rendus soient au moins aussi sûrs de fonctionnement que ceux d'applications traditionnelles.

Nos travaux traitent particulièrement de la détection d'anomalies dans les services cloud de type SaaS et PaaS. Nous nous sommes fixé quatre critères principaux pour la détection d'anomalies dans les services cloud : elle doit s'adapter aux changements de charge de travail et reconfiguration de services, elle doit se faire en ligne, de manière automatique, et avec un effort de configuration minimum en utilisant possiblement la même technique quelque soit le type de service.

Dans ce manuscrit, nous avons proposé une stratégie de détection qui repose sur le traitement de compteurs de performance par des techniques d'apprentissage automatique. Les différentes anomalies qu'il est possible de détecter sont les erreurs, les symptômes préliminaires de violations de service et les violations de service. Ces détections sont complémentaires. Dans le cas où une erreur n'est pas détectée, celle-ci peut potentiellement amener à une violation de service qui pourrait être évitée si la détection de ses symptômes préliminaires avait été quant à elle efficace. Dans le cas où cette dernière n'a pas été efficace, la détection de violations de service est le dernier niveau de détection qui peut permettre de lever une alarme.

L'évaluation de cette stratégie sur deux cas d'étude a été menée dans plusieurs buts et a permis de mettre en évidence plusieurs points :

— Identifier dans quelles mesures la détection d'anomalies de différents types était possible avec deux techniques proposées : une détection par apprentissage supervisé et une nouvelle technique de détection appelée DESC que nous avons définies dans nos travaux.
— Mettre en évidence le besoin d'utiliser plusieurs mesures de performance afin de s'assurer de la validation des performances de détection d'une technique.
— Identifier les tests à mener dans un but de validation rigoureuse des perfor-

mances de détection d'une technique. Des tests sur plusieurs jeux de données sont nécessaires afin de garantir ces performances en conditions opérationnelles.
— Montrer qu'il est possible d'effectuer un diagnostic sur l'origine d'une anomalie en même temps que la détection est réalisée.
— Montrer que la durée de validité des modèles de détection est nécessaire à prendre en considération pour généraliser les performances de détection d'une technique.
— Conclure que les performances de détection sont meilleures avec une source de monitoring au plus proche des composants d'un système (i.e. monitoring basé sur des compteurs de performance des SE des composants d'un système cible).
— Constater que les résultats de la technique de détection DESC hybride est prometteuse. Les résultats de DESC avec une analyse par fenêtre glissante n'ont pas été présentés dans ce manuscrit mais ont été publiés [Sauvanaud 2015]. Les résultats sont très prometteurs dans ce cas. Même si elle ne permet pas un aussi bon diagnostic d'anomalies que la technique par apprentissage supervisé implémentée avec l'algorithme Random Forest, DESC hybride donne des résultats similaires lorsque les performances de détection sont évaluées sur une prédiction qui a lieu un certain temps après l'entrainement des modèles. DESC possède également l'avantage d'une simplicité de déploiement et de configuration.
— Conclure que les performances de détection d'une stratégie de détection sont dépendantes du composant concerné et des anomalies présentes dans ces composants.
— Constater que la durée d'injection de fautes dans le but d'obtenir un jeu de données d'entrainement peut avoir un impact sur les performances de détection. Il est donc important d'établir une méthode précise définissant les caractéristiques d'une campagne d'injection supposée amener une représentation pertinente d'un système en comportement normal et en présence d'anomalie.
— Conclure qu'il existe un "dictionnaire d'anomalies" permettant d'identifier quelles anomalies subit un système, toutefois ce dictionnaire dépend du niveau d'anomalie (erreur, symptôme ou violation de service) et de l'application. Il ne peut donc pas être généralisé à tous types de service.

Perspectives

Dans nos prochains travaux, nous prévoyons de travailler sur la validation de notre stratégie de détection pour la détection de fautes au niveau de l'hyperviseur ainsi qu'au niveau applicatif des VMs d'un système cible.

Il est de plus important de continuer d'explorer les performances de détection de notre technique DESC. Nous souhaitons étudier davantage d'algorithmes de

clustering afin de mettre en œuvre la détection DESC avec un algorithme qui soit moins sensible aux variations locales de nos descripteurs de clusters. Cela peut être fait en utilisant de manière plus avancée l'algorithme Clustream qui gère la création de micro clusters pour les assembler en macro clusters, ou en utilisant une implémentation de clustering de sous-espaces de données tels que CLIQUE. Une autre piste est de changer la mesure de distance entre des observations permettant de créer des clusters. Nous utilisons actuellement la distance euclidienne. La distance de Mahalanobis serait un bon choix pour poursuivre nos travaux car elle prend en compte la variance des attributs ainsi que leur corrélation.

Concernant nos cas d'étude, l'étude d'un déploiement redondant de Clearwater permettrait de plus d'évaluer l'impact de la redondance sur les performances de détection de notre stratégie. Nous pourrions évaluer dans quelle mesure nos types de détection avec diagnostic sont toujours performants avec de la redondance.

Notre analyse des compteurs de performance utiles à la détection d'anomalies et d'une création d'un dictionnaire d'anomalies nous mène à vouloir évaluer dans quelle mesure les compteurs utiles à une détection dans un composant particulier peut être améliorée en tenant compte d'une analyse des logs du composant. Il serait de plus intéressant d'identifier quels compteurs de performances provenant du monitoring SE seraient susceptibles d'améliorer les performances de détection avec monitoring hyperviseur s'ils étaient ajoutés aux observations de monitoring hyperviseur. L'étude des moyens à mettre en œuvre pour acquérir ces compteurs à partir des hyperviseurs est alors à réaliser.

Enfin, la perspective majeure serait d'amener les modules de notre stratégie de détection à faire partie d'un benchmark de techniques de détection d'anomalies à partir de compteurs de performance. Le benchmark proposerait plusieurs techniques connues par apprentissage automatique ou statistiques (qui est l'autre grand domaine couvert par les techniques de détection d'anomalies). Les tests pourraient être menés de deux manières : 1) à partir d'une infrastructure compatible et d'expérimentations (avec lancements de campagnes d'injection dans le but d'obtenir des jeux de données labelisés), et 2) à partir d'un jeu de données avec des observations labelisées. Le benchmark proposerait différents modules de monitoring collectant des données de différentes sources tout en étant adaptables sur différentes infrastructures cloud (par exemple Amazon, VMware ou Openstack qui sont des solutions cloud populaires à l'heure de la rédaction de ce manuscrit) et différents modules d'injection de fautes (celui que nous présentons dans ce manuscrit suppose ne pas avoir accès au code d'une application testée mais nous en avons développé un qui fonctionne en modifiant les appels d'une application à certaines fonctions).

Le benchmark serait également paramétrable de manière à ce que des rapports de détection tels que les figures présentées dans nos deux derniers chapitres soient générés selon le besoin de l'utilisateur. L'utilisateur aurait dans ce cas la possibilité de lancer des tests, par exemple : "test de temps d'exécution d'un algorithme", "étude comparative d'algorithmes", ou "étude de sensibilité sur les paramètres d'une campagne d'injection".

Annexe A

Monitoring

A.1 Compteurs de performance en monitoring SE

A.1.1 Compteurs utilisés avec un apprentissage supervisé

1 bytes_in
2 bytes_out
3 cpu_aidle
4 cpu_idle
5 cpu_intr
6 cpu_nice
7 cpu_num
8 cpu_sintr
9 cpu_speed
10 cpu_steal
11 cpu_system
12 cpu_user
13 cpu_wio
14 disk_free
15 disk_free_absolute_rootfs
16 disk_free_percent_rootfs
17 disk_total
18 diskstat_sda_io_time
19 diskstat_sda_percent_io_time
20 diskstat_sda_read_bytes_per_sec
21 diskstat_sda_read_time
22 diskstat_sda_reads
23 diskstat_sda_reads_merged
24 diskstat_sda_weighted_io_time
25 diskstat_sda_write_bytes_per_sec
26 diskstat_sda_write_time
27 diskstat_sda_writes
28 diskstat_sda_writes_merged
29 entropy_avail
30 fault_injection
31 fault_intensity
32 fault_value
33 icmp_inaddrmaskreps
34 icmp_inaddrmasks
35 icmp_indestunreachs
36 icmp_inechoreps
37 icmp_inechos
38 icmp_inerrors
39 icmp_inmsgs
40 icmp_inparmprobs
41 icmp_inredirects
42 icmp_insrcquenchs
43 icmp_intimeexcds
44 icmp_intimestampreps
45 icmp_intimestamps
46 icmp_outaddrmaskreps
47 icmp_outaddrmasks
48 icmp_outdestunreachs
49 icmp_outechoreps
50 icmp_outechos
51 icmp_outerrors
52 icmp_outmsgs
53 icmp_outparmprobs
54 icmp_outredirects
55 icmp_outsrcquenchs
56 icmp_outtimeexcds
57 icmp_outtimestampreps
58 icmp_outtimestamps
59 io_busymax
60 io_max_svc_time
61 io_max_wait_time
62 io_nread
63 io_nwrite
64 io_reads
65 io_writes
66 ip_forwdatagrams
67 ip_fragcreates
68 ip_fragfails
69 ip_fragoks
70 ip_inaddrerrors
71 ip_indelivers
72 ip_indiscards
73 ip_inhdrerrors
74 ip_inreceives
75 ip_inunknownprotos
76 ip_outdiscards
77 ip_outnoroutes
78 ip_outrequests

79 ip_reasmfails
80 ip_reasmoks
81 ip_reasmreqds
82 ip_reasmtimeout
83 load_fifteen
84 load_five
85 load_one
86 mem_buffers
87 mem_cached
88 mem_dirty
89 mem_free
90 mem_hardware_corrupted
91 mem_mapped
92 mem_shared
93 mem_total
94 mem_writeback
95 part_max_used
96 pkts_in
97 pkts_out
98 proc_run
99 proc_total
100 procstat_gmond_cpu
101 procstat_gmond_mem
102 rx_bytes_eth0
103 rx_bytes_eth1
104 rx_bytes_lo
105 rx_drops_eth0
106 rx_drops_eth1
107 rx_drops_lo
108 rx_errs_eth0
109 rx_errs_eth1
110 rx_errs_lo
111 rx_pkts_eth0
112 rx_pkts_eth1
113 rx_pkts_lo
114 swap_free
115 swap_total
116 tcp_activeopens
117 tcp_attemptfails
118 tcp_currestab
119 tcp_estabresets
120 tcp_inerrs
121 tcp_insegs
122 tcp_outrsts
123 tcp_outsegs
124 tcp_passiveopens
125 tcp_retrans_percentage
126 tcp_retranssegs
127 tcpext_arpfilter
128 tcpext_delayedacklocked
129 tcpext_delayedacklost
130 tcpext_delayedacks
131 tcpext_embryonicrsts
132 tcpext_listendrops
133 tcpext_listenoverflows
134 tcpext_lockdroppedicmps
135 tcpext_ofopruned
136 tcpext_outofwindowicmps
137 tcpext_pawsactive
138 tcpext_pawsestab
139 tcpext_pawspassive
140 tcpext_prunecalled
141 tcpext_rcvpruned
142 tcpext_syncookiesfailed
143 tcpext_syncookiesrecv
144 tcpext_syncookiessent
145 tcpext_tcpabortfailed
146 tcpext_tcpabortonclose
147 tcpext_tcpabortondata
148 tcpext_tcpabortonlinger
149 tcpext_tcpabortonmemory
150 tcpext_tcpabortonsyn
151 tcpext_tcpabortontimeout
152 tcpext_tcpbacklogdrop
153 tcpext_tcpdeferacceptdrop
154 tcpext_tcpdirectcopyfrombacklog
155 tcpext_tcpdirectcopyfromprequeue
156 tcpext_tcpdsackignorednoundo
157 tcpext_tcpdsackignoredold
158 tcpext_tcpdsackoforecv
159 tcpext_tcpdsackofosent
160 tcpext_tcpdsackoldsent
161 tcpext_tcpdsackrecv
162 tcpext_tcpdsackundo
163 tcpext_tcpfackreorder
164 tcpext_tcpfastretrans
165 tcpext_tcpforwardretrans
166 tcpext_tcpfullundo
167 tcpext_tcphpacks
168 tcpext_tcphphits
169 tcpext_tcphphitstouser
170 tcpext_tcploss
171 tcpext_tcploss_percentage
172 tcpext_tcplossfailures
173 tcpext_tcplossundo
174 tcpext_tcplostretransmit
175 tcpext_tcpmd5notfound
176 tcpext_tcpmd5unexpected
177 tcpext_tcpmemorypressures
178 tcpext_tcpminttldrop
179 tcpext_tcppartialundo
180 tcpext_tcpprequeued
181 tcpext_tcpprequeuedropped
182 tcpext_tcppureacks
183 tcpext_tcprcvcollapsed
184 tcpext_tcprenofailures
185 tcpext_tcprenorecovery
186 tcpext_tcprenorecoveryfail
187 tcpext_tcprenoreorder
188 tcpext_tcpsackdiscard
189 tcpext_tcpsackfailures
190 tcpext_tcpsackmerged
191 tcpext_tcpsackrecovery
192 tcpext_tcpsackrecoveryfail
193 tcpext_tcpsackreneging
194 tcpext_tcpsackreorder
195 tcpext_tcpsackshifted
196 tcpext_tcpsackshiftfallback
197 tcpext_tcpschedulerfailed
198 tcpext_tcpslowstartretrans
199 tcpext_tcpspuriousrtos
200 tcpext_tcptimeouts
201 tcpext_tcptsreorder
202 tcpext_tw

203 tcpext_twkilled
204 tcpext_twrecycled
205 tx_bytes_eth0
206 tx_bytes_eth1
207 tx_bytes_lo
208 tx_drops_eth0
209 tx_drops_eth1
210 tx_drops_lo
211 tx_errs_eth0
212 tx_errs_eth1
213 tx_errs_lo
214 tx_pkts_eth0
215 tx_pkts_eth1
216 tx_pkts_lo
217 udp_indatagrams
218 udp_inerrors
219 udp_noports
220 udp_outdatagrams
221 udp_rcvbuferrors
222 udp_sndbuferrors
223 vm_kswapd_skip_congestion_wait
224 vm_pgmajfault
225 vm_pgpgin
226 vm_pgpgout
227 vm_vmeff

A.1.2 Compteurs utilisés avec la détection DESC

A.1.2.1 Sous-flux CPU

1 cpu_aidle
2 cpu_idle
2 cpu_nice
3 cpu_num
4 cpu_speed
5 cpu_system
6 cpu_user
7 cpu_wio

A.1.2.2 Sous-flux mémoire

1 mem_buffers
2 mem_cached
3 mem_dirty
4 mem_free
5 mem_hardware_corrupted
6 mem_mapped
7 mem_shared
8 mem_total
9 mem_writeback

A.1.2.3 Sous-flux disque

1 disk_free
2 disk_total
3 diskstat_sda_io_time
4 diskstat_sda_percent_io_time
5 diskstat_sda_read_bytes_per_sec
6 diskstat_sda_read_time
7 diskstat_sda_reads
8 diskstat_sda_reads_merged
9 diskstat_sda_weighted_io_time
10 diskstat_sda_write_bytes_per_sec
11 diskstat_sda_write_time
12 diskstat_sda_writes
13 diskstat_sda_writes_merged

A.1.2.4 Sous-flux réseau

1 tcp_insegs
2 tcp_outsegs
3 tcp_retrans_percentage
4 tcpext_tcpsackfailures

5 tcpext_tcpslowstartretrans
6 tcpext_tcptimeouts

A.2 Compteurs de performance en monitoring hyperviseur

1 cpu_costop
2 cpu_costop_1
3 cpu_costop
4 cpu_demand
5 cpu_entitlement
6 cpu_idle
7 cpu_idle_1
8 cpu_idle
9 cpu_latency
10 cpu_maxlimited
11 cpu_maxlimited_1
12 cpu_maxlimited
13 cpu_overlap
14 cpu_overlap_1
15 cpu_overlap
16 cpu_ready
17 cpu_ready_1
18 cpu_ready
19 cpu_run
20 cpu_run_1
21 cpu_run
22 cpu_swapwait
23 cpu_swapwait_1
24 cpu_swapwait
25 cpu_system
26 cpu_system_1
27 cpu_system
28 cpu_used
29 cpu_used_1
30 cpu_used
31 cpu_wait
32 cpu_wait_1
33 cpu_wait
34 datastore_maxTotalLatency
35 datastore_numberReadAveraged
36 datastore_numberWriteAveraged
37 datastore_read
38 datastore_totalReadLatency
39 datastore_totalWriteLatency
40 datastore_write
41 disk_busResets_naa
42 disk_commandsAborted_naa
43 disk_commandsAveraged_naa
44 disk_commands_naa
45 disk_maxTotalLatency
46 disk_numberReadAveraged_naa
47 disk_numberRead_naa
48 disk_numberWriteAveraged_naa
49 disk_numberWrite_naa
50 disk_read
51 disk_read_naa
52 disk_write
53 disk_write_naa
54 mem_activewrite
55 mem_compressed
56 mem_compressionRate
57 mem_decompressionRate
58 mem_entitlement
59 mem_latency
60 mem_llSwapInRate
61 mem_llSwapOutRate
62 mem_overheadMax
63 mem_overheadTouched
64 mem_swapinRate
65 mem_swapoutRate
66 mem_zipped
67 mem_zipSaved
68 net_broadcastRx_nic1
69 net_broadcastRx_nic2
70 net_broadcastRx
71 net_broadcastTx_nic1
72 net_broadcastTx_nic2
73 net_broadcastTx
74 net_bytesRx
75 net_bytesRx_nic1
76 net_bytesRx_nic2
77 net_bytesTx
78 net_bytesTx_nic1
79 net_bytesTx_nic2
80 net_droppedRx_nic1
81 net_droppedRx_nic2
82 net_droppedRx
83 net_droppedTx_nic1
84 net_droppedTx_nic2
85 net_droppedTx
86 net_multicastRx_nic1
87 net_multicastRx_nic2
88 net_multicastRx
89 net_multicastTx_nic1
90 net_multicastTx_nic2
91 net_multicastTx
92 net_packetsRx_nic1
93 net_packetsRx_nic2
94 net_packetsRx
95 net_packetsTx_nic1
96 net_packetsTx_nic2
97 net_packetsTx
98 net_received
99 net_received_nic1
100 net_received_nic2
101 net_transmitted
102 net_transmitted_nic1
103 net_transmitted_nic2
104 power_energy
105 power_power
106 rescpu_actav15
107 rescpu_actav1
108 rescpu_actav5

109 rescpu_actpk15
110 rescpu_actpk1
111 rescpu_actpk5
112 rescpu_maxLimited15
113 rescpu_maxLimited1
114 rescpu_maxLimited5
115 rescpu_runav15
116 rescpu_runav1
117 rescpu_runav5
118 rescpu_runpk15
119 rescpu_runpk1
120 rescpu_runpk5
121 rescpu_sampleCount
122 rescpu_samplePeriod
123 sys_heartbeat
124 sys_uptime
125 virtualDisk_largeSeeks_scsi
126 virtualDisk_mediumSeeks_scsi
127 virtualDisk_numberReadAveraged_scsi
128 virtualDisk_numberWriteAveraged_scsi
129 virtualDisk_readIOSize_scsi
130 virtualDisk_readLatencyUS_scsi
131 virtualDisk_readLoadMetric_scsi
132 virtualDisk_readOIO_scsi
133 virtualDisk_read_scsi
134 virtualDisk_smallSeeks_scsi
135 virtualDisk_totalReadLatency_scsi
136 virtualDisk_totalWriteLatency_scsi
137 virtualDisk_writeIOSize_scsi
138 virtualDisk_writeLatencyUS_scsi
139 virtualDisk_writeLoadMetric_scsi
140 virtualDisk_writeOIO_scsi
141 virtualDisk_write_scsi
142 Memctl?
143 Memctl_MBytes
144 Memctl_Target_MBytes
145 Memctl_Max_MBytes
146 Swapped_MBytes
147 Swap_Target_MBytes
148 Swap_Read_MBytes/sec
149 Swap_Written_MBytes/sec
150 Overhead_UW_MBytes
151 Overhead_MBytes
152 Overhead_Max_MBytes

Annexe B

Appel SIPp pour le cas d'étude Clearwater

Listing B.1 – Appel entre deux utilisateurs simulés par le benchmark SIPp.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- This program is free software; you can redistribute it and/or -->
<!-- modify it under the terms of the GNU General Public License as -->
<!-- published by the Free Software Foundation; either version 2 of the -->
<!-- License, or (at your option) any later version. -->
<!-- -->
<!-- This program is distributed in the hope that it will be useful, -->
<!-- but WITHOUT ANY WARRANTY; without even the implied warranty of -->
<!-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the -->
<!-- GNU General Public License for more details. -->
<!-- -->
<!-- You should have received a copy of the GNU General Public License -->
<!-- along with this program; if not, write to the -->
<!-- Free Software Foundation, Inc., -->
<!-- 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA -->
<!-- -->

<scenario name="Call Load Test">

  <User variables="my_dn,peer_dn,call_repeat" />
  <nop hide="true">
    <action>
      <!-- Get my and peer's DN -->
      <assignstr assign_to="my_dn" value="[field0]" />
      <!-- field1 is my_auth, but we can't store it in a variable -->
      <assignstr assign_to="peer_dn" value="[field2]" />
      <!-- field3 is peer_auth, but we can't store it in a variable -->
      <assign assign_to="reg_repeat" value="0"/>
      <assign assign_to="call_repeat" value="0"/>
    </action>
  </nop>

  <pause distribution="uniform" min="0" max="5000"/>

  <send>
    <![CDATA[

      REGISTER sip:[$my_dn]@[service] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];rport;branch=[branch]-
        [$my_dn]-[$reg_repeat]
      Route: <sip:[service];transport=[transport];lr>
      Max-Forwards: 70
      From: <sip:[$my_dn]@[service]>;tag=[pid]SIPpTag00[call_number]
      To: <sip:[$my_dn]@[service]>
      Call-ID: [$my_dn]///[call_id]
      CSeq: [cseq] REGISTER
      User-Agent: Accession 4.0.0.0
      Supported: outbound, path
      Contact: <sip:[$my_dn]@[local_ip]:[local_port];transport=[transport];ob>;
        +sip.ice;reg-id=1;+sip.instance=
```

```
      "<urn:uuid:00000000-0000-0000-0000-000000000001>"
    Expires: 3600
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
      MESSAGE, OPTIONS
    Content-Length: 0

  ]]>
</send>

<recv response="401" auth="true">
  <action>
    <add assign_to="reg_repeat" value="1" />
  </action>
</recv>

<send>
  <![CDATA[

    REGISTER sip:[$my_dn]@[service] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];rport;branch=[branch]-
      [$my_dn]-[$reg_repeat]
    Route: <sip:[service];transport=[transport];lr>
    Max-Forwards: 70
    From: <sip:[$my_dn]@[service]>;tag=[pid]SIPpTag00[call_number]
    To: <sip:[$my_dn]@[service]>
    Call-ID: [$my_dn]///[call_id]
    CSeq: [cseq] REGISTER
    User-Agent: Accession 4.0.0.0
    Supported: outbound, path
    Contact: <sip:[$my_dn]@[local_ip]:[local_port];transport=[transport];ob>;
      +sip.ice;reg-id=1;+sip.instance=
      "<urn:uuid:00000000-0000-0000-0000-000000000001>"
    Expires: 3600
    [field1]
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
      MESSAGE, OPTIONS
    Content-Length: 0

  ]]>
</send>

<recv response="200">
  <action>
    <ereg regexp="rport=([^;]*);.*received=([^;]*);" search_in="hdr"
      header="Via:" assign_to="dummy,nat_port,nat_ip_addr" />
    <add assign_to="reg_repeat" value="1" />
  </action>
</recv>
<Reference variables="dummy" />

<send>
  <![CDATA[

    REGISTER sip:[$peer_dn]@[service] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];rport;branch=[branch]-
      [$peer_dn]-[$reg_repeat]
    Route: <sip:[service];transport=[transport];lr>
    Max-Forwards: 70
    From: <sip:[$peer_dn]@[service]>;tag=[pid]SIPpTag00[call_number]
    To: <sip:[$peer_dn]@[service]>
    Call-ID: [$peer_dn]///[call_id]
    CSeq: [cseq] REGISTER
    User-Agent: Accession 4.0.0.0
    Supported: outbound, path
    Contact: <sip:[$peer_dn]@[local_ip]:[local_port];transport=[transport];
      ob>;+sip.ice;reg-id=1;+sip.instance=
      "<urn:uuid:00000000-0000-0000-0000-000000000001>"
    Expires: 3600
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
      MESSAGE, OPTIONS
    Content-Length: 0

  ]]>
</send>

<recv response="401" auth="true">
  <action>
    <add assign_to="reg_repeat" value="1" />
  </action>
</recv>

<send>
  <![CDATA[
```

```
    REGISTER sip:[$peer_dn]@[service] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];rport;branch=[branch]-
      [$peer_dn]-[$reg_repeat]
    Route: <sip:[service];transport=[transport];lr>
    Max-Forwards: 70
    From: <sip:[$peer_dn]@[service]>;tag=[pid]SIPpTag00[call_number]
    To: <sip:[$peer_dn]@[service]>
    Call-ID: [$peer_dn]///[call_id]
    CSeq: [cseq] REGISTER
    User-Agent: Accession 4.0.0.0
    Supported: outbound, path
    Contact: <sip:[$peer_dn]@[local_ip]:[local_port];transport=[transport];
      ob>;+sip.ice;reg-id=1;+sip.instance=
      "<urn:uuid:00000000-0000-0000-0000-000000000001>"
    Expires: 3600
    [field3]
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
      MESSAGE, OPTIONS
    Content-Length: 0

  ]]>
</send>

<recv response="200" >
  <action>
    <add assign_to="reg_repeat" value="1" />
  </action>
</recv>

<label id="1"/>

<send start_rtd="register">
  <![CDATA[

    REGISTER sip:[$my_dn]@[service] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];rport;branch=[branch]-
      [$my_dn]-[$reg_repeat]
    Route: <sip:[service];transport=[transport];lr>
    Max-Forwards: 70
    From: <sip:[$my_dn]@[service]>;tag=[pid]SIPpTag00[call_number]
    To: <sip:[$my_dn]@[service]>
    Call-ID: [$my_dn]///[call_id]
    CSeq: [cseq] REGISTER
    User-Agent: Accession 4.0.0.0
    Supported: outbound, path
    Contact: <sip:[$my_dn]@[$nat_ip_addr]:[$nat_port];transport=[transport];
      ob>;+sip.ice;reg-id=1;+sip.instance=
      "<urn:uuid:00000000-0000-0000-0000-000000000001>"
    Contact: <sip:[$my_dn]@[local_ip]:[local_port];transport=[transport];
      ob>;expires=0;+sip.ice;reg-id=1;+sip.instance="<urn:uuid:00000000-0000-0000-0000-000000000001>"
    Expires: 3600
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
      MESSAGE, OPTIONS
    Content-Length: 0

  ]]>
</send>

<recv response="200" rtd="register">
  <action>
    <add assign_to="reg_repeat" value="1" />
  </action>
</recv>

<send start_rtd="register">
  <![CDATA[

    REGISTER sip:[$peer_dn]@[service] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];rport;branch=[branch]-
      [$my_dn]-[$reg_repeat]
    Route: <sip:[service];transport=[transport];lr>
    Max-Forwards: 70
    From: <sip:[$peer_dn]@[service]>;tag=[pid]SIPpTag00[call_number]
    To: <sip:[$peer_dn]@[service]>
    Call-ID: [$my_dn]///[call_id]
    CSeq: [cseq] REGISTER
    User-Agent: Accession 4.0.0.0
    Supported: outbound, path
    Contact: <sip:[$peer_dn]@[$nat_ip_addr]:[$nat_port];transport=[transport];
      ob>;+sip.ice;reg-id=1;+sip.instance="<urn:uuid:00000000-0000-0000-0000-000000000001>"
    Contact: <sip:[$peer_dn]@[local_ip]:[local_port];transport=[transport];
      ob>;expires=0;+sip.ice;reg-id=1;+sip.instance="<urn:uuid:00000000-0000-0000-0000-000000000001>"
    Expires: 3600
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
```

```
      MESSAGE, OPTIONS
    Content-Length: 0

  ]]>
</send>

<recv response="200" rtd="register" >
  <action>
    <add assign_to="reg_repeat" value="1" />
  </action>
</recv>

<label id="2" />

<nop hide="true">
  <action>
    <add assign_to="call_repeat" value="1" />
    <sample assign_to="pre_call_delay" distribution="uniform" min="0"
      max="5000" />
    <assign assign_to="post_call_delay" value="5000" />
    <subtract assign_to="post_call_delay" variable="pre_call_delay" />
  </action>
</nop>


<send start_rtd="call-setup">
  <![CDATA[

    INVITE sip:[$peer_dn]@[service] SIP/2.0
    Via: SIP/2.0/[transport] [$nat_ip_addr]:[$nat_port];rport;branch=z9hG4bK-
      [$my_dn]-[call_number]-[$call_repeat]-1
    Max-Forwards: 70
    From: sip:[$my_dn]@[service];tag=[pid]SIPpTag00[call_number]1234
    To: sip:[$peer_dn]@[service]
    Contact: <sip:[$my_dn]@[$nat_ip_addr]:[$nat_port];transport=[transport];
      ob>;+sip.ice
    Call-ID: [$my_dn]-[$call_repeat]///[call_id]
    CSeq: [cseq] INVITE
    Route: <sip:[service];transport=[transport];lr>
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
      MESSAGE, OPTIONS
    Supported: replaces, 100rel, timer, norefersub
    Session-Expires: 1800
    Min-SE: 90
    User-Agent: Accession 4.0.0.0
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=- 3547439529 3547439529 IN IP4 23.23.222.251
    s=pjmedia
    c=IN IP4 23.23.222.251
    b=AS:84
    t=0 0
    a=X-nat:3
    m=audio 34012 RTP/AVP 120 121 106 0 8 96
    c=IN IP4 23.23.222.251
    b=TIAS:64000
    a=rtcp:41203 IN IP4 23.23.222.251
    a=sendrecv
    a=rtpmap:120 SILK/8000
    a=fmtp:120 maxaveragebitrate=64000;useinbandfec=1;usedtx=1
    a=rtpmap:121 SILK/16000
    a=fmtp:121 maxaveragebitrate=64000;useinbandfec=1;usedtx=1
    a=rtpmap:106 AMR-WB/16000
    a=rtpmap:0 PCMU/8000
    a=rtpmap:8 PCMA/8000
    a=rtpmap:96 telephone-event/8000
    a=fmtp:96 0-15
    a=ice-ufrag:63eab89f
    a=ice-pwd:2cbaece5
    a=candidate:Sc0a801c2 1 UDP 1694498815 192.91.191.20 54989 typ srflx raddr
      192.168.1.194 rport 42506
    a=candidate:Hc0a801c2 1 UDP 2130706431 192.168.1.194 42506 typ host
    a=candidate:R1717defb 1 UDP 16777215 23.23.222.251 34012 typ relay raddr
      192.91.191.20 rport 27564
    a=candidate:Sc0a801c2 2 UDP 1694498814 192.91.191.20 40944 typ srflx raddr
      192.168.1.194 rport 59288
    a=candidate:Hc0a801c2 2 UDP 2130706430 192.168.1.194 59288 typ host
    a=candidate:R1717defb 2 UDP 16777214 23.23.222.251 41203 typ relay raddr
      192.91.191.20 rport 49972

  ]]>
</send>
```

```
<!-- The 100 response is generated by sprout and the proxied INVITE can be
     received in either order. -->
<!-- If we receive the 100 response first, move onto label 3 where we expect
     the INVITE and then continue. -->
<!-- If we receive the INVITE first, expect a 100 response and then move onto
    label 4, where we continue. -->
<recv response="100" optional="true" next="3">
</recv>

<recv request="INVITE">
</recv>

<nop hide="true">
  <action>
    <assignstr assign_to="uas_via" value="[last_Via:]" />
  </action>
</nop>

<recv response="100" next="4">
</recv>

<label id="3" />

<recv request="INVITE">
</recv>

<nop hide="true">
  <action>
    <assignstr assign_to="uas_via" value="[last_Via:]" />
  </action>
</nop>

<label id="4" />

<send>
  <![CDATA[

    SIP/2.0 100 Trying
    [$uas_via]
    [last_Record-Route:]
    Call-ID: [$my_dn]-[$call_repeat]///[call_id]
    From: <sip:[$peer_dn]@[service]>;tag=[pid]SIPpTag00[call_number]1234
    To: <sip:[$my_dn]@[service]>
    [last_CSeq:]
    Content-Length: 0

  ]]>
</send>

<send>
  <![CDATA[

    SIP/2.0 180 Ringing
    [$uas_via]
    [last_Record-Route:]
    Call-ID: [$my_dn]-[$call_repeat]///[call_id]
    From: <sip:[$peer_dn]@[service]>;tag=[pid]SIPpTag00[call_number]1234
    To: <sip:[$my_dn]@[service]>;tag=[pid]SIPpTag00[call_number]4321
    [last_CSeq:]
    Contact: <sip:[$peer_dn]@[$nat_ip_addr]:[$nat_port];transport=[transport];
      ob>;+sip.ice
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
      MESSAGE, OPTIONS
    Content-Length: 0

  ]]>
</send>

<recv response="180">
</recv>

<send>
  <![CDATA[

    SIP/2.0 200 OK
    [$uas_via]
    [last_Record-Route:]
    Call-ID: [$my_dn]-[$call_repeat]///[call_id]
    From: <sip:[$peer_dn]@[service]>;tag=[pid]SIPpTag00[call_number]1234
    To: <sip:[$my_dn]@[service]>;tag=[pid]SIPpTag00[call_number]4321
    [last_CSeq:]
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
      MESSAGE, OPTIONS
```

```
    Contact: <sip:[$peer_dn]@[$nat_ip_addr]:[$nat_port];transport=[transport];
      ob>;+sip.ice
    Supported: replaces, 100rel, timer, norefersub
    Session-Expires: 1800;refresher=uac
    Content-Type: application/sdp
    Content-Length: 948

    v=0
    o=- 3547439528 3547439529 IN IP4 23.23.222.251
    s=pjmedia
    c=IN IP4 23.23.222.251
    b=AS:84
    t=0 0
    a=X-nat:3
    m=audio 59808 RTP/AVP 120 96
    c=IN IP4 23.23.222.251
    b=TIAS:64000
    a=rtcp:36110 IN IP4 23.23.222.251
    a=sendrecv
    a=rtpmap:120 SILK/8000
    a=fmtp:120 maxaveragebitrate=64000;useinbandfec=1;usedtx=1
    a=rtpmap:96 telephone-event/8000
    a=fmtp:96 0-15
    a=ice-ufrag:32ffb0d4
    a=ice-pwd:6b7c406b
    a=candidate:Sac123cc4 1 UDP 1694498815 192.91.191.29 57173 typ srflx raddr
      172.18.60.196 rport 57931
    a=candidate:Hac123cc4 1 UDP 2130706431 172.18.60.196 57931 typ host
    a=candidate:R1717defb 1 UDP 16777215 23.23.222.251 59808 typ relay raddr
      192.91.191.29 rport 62536
    a=candidate:Sac123cc4 2 UDP 1694498814 192.91.191.29 60865 typ srflx raddr
      172.18.60.196 rport 59842
    a=candidate:Hac123cc4 2 UDP 2130706430 172.18.60.196 59842 typ host
    a=candidate:R1717defb 2 UDP 16777214 23.23.222.251 36110 typ relay raddr
      192.91.191.29 rport 57129

  ]]>
</send>

<recv response="200" rrs="true">
</recv>

<send>
  <![CDATA[

    ACK sip:[$peer_dn]@[$nat_ip_addr]:[$nat_port];transport=[transport];
      ob SIP/2.0
    Via: SIP/2.0/[transport] [$nat_ip_addr]:[$nat_port];rport;branch=z9hG4bK-
      [$my_dn]-[call_number]-[$call_repeat]-1
    [routes]
    Max-Forwards: 70
    From: sip:[$my_dn]@[service];tag=[pid]SIPpTag00[call_number]1234
    To: sip:[$peer_dn]@[service];tag=[pid]SIPpTag00[call_number]4321
    Call-ID: [$my_dn]-[$call_repeat]///[call_id]
    CSeq: [cseq] ACK
    Content-Length: 0

  ]]>
</send>

<recv request="ACK">
</recv>

<send>
  <![CDATA[

    UPDATE sip:[$peer_dn]@[$nat_ip_addr]:[$nat_port];transport=[transport];
      ob SIP/2.0
    Via: SIP/2.0/[transport] [$nat_ip_addr]:[$nat_port];rport;branch=z9hG4bK-
      [$my_dn]-[call_number]-[$call_repeat]-2
    [routes]
    Max-Forwards: 70
    From: sip:[$my_dn]@[service];tag=[pid]SIPpTag00[call_number]1234
    To: sip:[$peer_dn]@[service];tag=[pid]SIPpTag00[call_number]4321
    Contact: <sip:[$my_dn]@[$nat_ip_addr]:[$nat_port];transport=[transport];
      ob>;+sip.ice
    Call-ID: [$my_dn]-[$call_repeat]///[call_id]
    CSeq: [cseq] UPDATE
    Session-Expires: 1800;refresher=uac
    Min-SE: 90
    User-Agent: Accession 4.0.0.0
    Content-Type: application/sdp
    Content-Length: 843
```

```
    v=0
    o=- 3547439529 3547439530 IN IP4 23.23.222.251
    s=pjmedia
    c=IN IP4 192.91.191.20
    b=AS:84
    t=0 0
    a=X-nat:3
    m=audio 54989 RTP/AVP 120 121 106 0 8 96
    c=IN IP4 192.91.191.20
    b=TIAS:64000
    a=sendrecv
    a=rtpmap:120 SILK/8000
    a=fmtp:120 maxaveragebitrate=64000;useinbandfec=1;usedtx=1
    a=rtpmap:121 SILK/16000
    a=fmtp:121 maxaveragebitrate=64000;useinbandfec=1;usedtx=1
    a=rtpmap:106 AMR-WB/16000
    a=rtpmap:0 PCMU/8000
    a=rtpmap:8 PCMA/8000
    a=rtpmap:96 telephone-event/8000
    a=fmtp:96 0-15
    a=ice-ufrag:63eab89f
    a=ice-pwd:2cbaece5
    a=rtcp:40944 IN IP4 192.91.191.20
    a=candidate:Sc0a801c2 1 UDP 1694498815 192.91.191.20 54989 typ srflx raddr
      192.168.1.194 rport 42506
    a=candidate:Sc0a801c2 2 UDP 1694498814 192.91.191.20 40944 typ srflx raddr
      192.168.1.194 rport 59288
    a=remote-candidates:1 23.23.222.251 59808 2 23.23.222.251 36110

  ]]>
</send>

<recv request="UPDATE">
</recv>

<send>
  <![CDATA[

    SIP/2.0 200 OK
    [last_Via:]
    [last_Record-Route:]
    Call-ID: [$my_dn]-[$call_repeat]///[call_id]
    From: sip:[$peer_dn]@[service];tag=[pid]SIPpTag00[call_number]1234
    To: sip:[$my_dn]@[service];tag=[pid]SIPpTag00[call_number]4321
    [last_CSeq:]
    Session-Expires: 1800;refresher=uac
    Contact: <sip:[$peer_dn]@[$nat_ip_addr][$nat_port];transport=[transport];
      ob>;+sip.ice
    Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, SUBSCRIBE, NOTIFY, REFER,
      MESSAGE, OPTIONS
    Supported: replaces, 100rel, timer, norefersub
    Content-Type: application/sdp
    Content-Length: 606

    v=0
    o=- 3547439528 3547439530 IN IP4 23.23.222.251
    s=pjmedia
    c=IN IP4 23.23.222.251
    b=AS:84
    t=0 0
    a=X-nat:3
    m=audio 59808 RTP/AVP 120 96
    c=IN IP4 23.23.222.251
    b=TIAS:64000
    a=sendrecv
    a=rtpmap:120 SILK/8000
    a=fmtp:120 maxaveragebitrate=64000;useinbandfec=1;usedtx=1
    a=rtpmap:96 telephone-event/8000
    a=fmtp:96 0-15
    a=ice-ufrag:32ffb0d4
    a=ice-pwd:6b7c406b
    a=rtcp:36110 IN IP4 23.23.222.251
    a=candidate:R1717defb 1 UDP 16777215 23.23.222.251 59808 typ relay raddr
      192.91.191.29 rport 62536
    a=candidate:R1717defb 2 UDP 16777214 23.23.222.251 36110 typ relay raddr
      192.91.191.29 rport 57129

  ]]>
</send>

<recv response="200" rtd="call-setup">
</recv>
```

```
  <send start_rtd="call-teardown">
    <![CDATA[

      BYE sip:[$peer_dn]@[$nat_ip_addr]:[$nat_port];transport=[transport];ob
        SIP/2.0
      Via: SIP/2.0/[transport] [$nat_ip_addr]:[$nat_port];rport;branch=z9hG4bK-
        [$my_dn]-[call_number]-[$call_repeat]-3
      [routes]
      From: <sip:[$my_dn]@[service]>;tag=[pid]SIPpTag00[call_number]1234
      To: <sip:[$peer_dn]@[service]>;tag=[pid]SIPpTag00[call_number]4321
      Call-ID: [$my_dn]-[$call_repeat]///[call_id]
      CSeq: [cseq] BYE
      Contact: <sip:[$my_dn]@[$nat_ip_addr][$nat_port];transport=[transport];
        ob>;+sip.ice
      Max-Forwards: 70
      Subject: Performance Test
      Content-Length: 0

    ]]>
  </send>

  <recv request="BYE">
  </recv>

  <send>
    <![CDATA[

      SIP/2.0 200 OK
      [last_Via:]
      [last_Record-Route:]
      From: sip:[$peer_dn]@[service];tag=[pid]SIPpTag00[call_number]1234
      To: sip:[$my_dn]@[service];tag=[pid]SIPpTag00[call_number]4321
      Call-ID: [$my_dn]-[$call_repeat]///[call_id]
      [last_CSeq:]
      Contact: <sip:[$peer_dn]@[$nat_ip_addr][$nat_port];transport=[transport];
        ob>;+sip.ice
      Content-Length: 0

    ]]>
  </send>

  <recv response="200" rtd="call-teardown">
  </recv>

  <!-- definition of the response time repartition table (unit is ms) -->
  <ResponseTimeRepartition value="500, 1000, 2000, 3000, 4000, 7000, 10000"/>

  <!-- definition of the call length repartition table (unit is ms) -->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>
```

Bibliographie

[Aggarwal 2003] Charu C. Aggarwal, Jiawei Han, Jianyong Wang et Philip S. Yu. *A Framework for Clustering Evolving Data Streams.* In Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29, VLDB '03, pages 81–92. VLDB Endowment, 2003. (Cité en pages 28 et 48.)

[Aggarwal 2004] Charu C. Aggarwal, Jiawei Han, Jianyong Wang et Philip S. Yu. *A Framework for Projected Clustering of High Dimensional Data Streams.* In Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB '04, pages 852–863. VLDB Endowment, 2004. (Cité en page 28.)

[Aggarwal 2013a] Charu C. Aggarwal. *A survey of stream clustering algorithms.* In Data Clustering : Algorithms and Applications, pages 231–258, 2013. (Cité en page 28.)

[Aggarwal 2013b] Charu C. Aggarwal et Chandan K. Reddy. Data clustering : Algorithms and applications. Chapman & Hall/CRC, 1st édition, 2013. (Cité en page 27.)

[Agrawal 1998] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos et Prabhakar Raghavan. *Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications.* SIGMOD Rec., vol. 27, no. 2, pages 94–105, Juin 1998. (Cité en pages 27 et 49.)

[Aleskerov 1997] E. Aleskerov, B. Freisleben et B. Rao. *CARDWATCH : a neural network based database mining system for credit card fraud detection.* In Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997, pages 220–226, Mar 1997. (Cité en page 25.)

[Allix 2016] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, Radu State et Yves Le Traon. *Empirical assessment of machine learning-based malware detectors for Android.* Empirical Software Engineering, vol. 21, no. 1, pages 183–211, 2016. (Cité en page 34.)

[Alpaydin 2004] Ethem Alpaydin. Introduction to machine learning (adaptive computation and machine learning). The MIT Press, 2004. (Cité en page 23.)

[Arlat 1990] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins et D. Powell. *Fault injection for dependability validation : a*

methodology and some applications. IEEE Transactions on Software Engineering, vol. 16, no. 2, pages 166–182, 1990. (Cité en page 30.)

[Avizienis 2004] A. Avizienis, J.-C. Laprie, B. Randell et C. Landwehr. *Basic concepts and taxonomy of dependable and secure computing.* IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pages 11–33, 2004. (Cité en pages 1, 13 et 63.)

[Bauer 2012] Eric Bauer et Randee Adams. Reliability and availability of cloud computing. Wiley-IEEE Press, 1st édition, 2012. (Cité en page 9.)

[Berral 2010] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà et Jordi Torres. *Towards Energy-aware Scheduling in Data Centers Using Machine Learning.* In Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking, e-Energy '10, pages 215–224, New York, NY, USA, 2010. ACM. (Cité en page 20.)

[Beyer 1999] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan et Uri Shaft. *When Is "Nearest Neighbor" Meaningful ?* In Catriel Beeri et Peter Buneman, editeurs, Database Theory — ICDT'99, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer Berlin Heidelberg, 1999. (Cité en pages 27 et 45.)

[Bhat 2013] Amjad Hussain Bhat, Sabyasachi Patra et Debasish Jena. *Machine learning approach for intrusion detection on cloud virtual machines.* International Journal of Application or Innovation in Engineering & Management (IJAIEM), vol. 2, no. 6, pages 56–66, 2013. (Cité en page 20.)

[Bodík 2009] Peter Bodík, Rean Griffith, Charles Sutton, Armando Fox, Michael Jordan et David Patterson. *Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters.* In Proceedings of the 2009 Conference on Hot Topics in Cloud Computing, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association. (Cité en page 20.)

[Bradley 1997] Andrew P Bradley. *The use of the area under the ROC curve in the evaluation of machine learning algorithms.* Pattern recognition, vol. 30, no. 7, pages 1145–1159, 1997. (Cité en page 33.)

[Breiman 2001] Leo Breiman. *Random Forests.* Machine Learning, vol. 45, no. 1, pages 5–32, 2001. (Cité en pages 46 et 71.)

[Buzen 1973] J. P. Buzen et U. O. Gagliardi. *The Evolution of Virtual Machine Architecture.* In Proceedings of the June 4-8, 1973, National Computer Conference and Exposition, AFIPS '73, pages 291–299, New York, NY, USA, 1973. ACM. (Cité en page 11.)

[Cao 2006] Feng Cao, Martin Ester, Weining Qian et Aoying Zhou. *Density-based clustering over an evolving data stream with noise.* In In 2006 SIAM Conference on Data Mining, pages 328–339, 2006. (Cité en pages 28 et 48.)

[Cassisi 2013] Carmelo Cassisi, Alfredo Ferro, Rosalba Giugno, Giuseppe Pigola et Alfredo Pulvirenti. *Enhancing Density-based Clustering : Parameter Reduc-*

tion and Outlier Detection. Inf. Syst., vol. 38, no. 3, pages 317–330, Mai 2013. (Cité en page 27.)

[Cerf 2016] Sophie Cerf, Mihaly Berekmeri, Bogdan Robu, Nicolas Marchand et Sara Bouchenak. *Cost Function based Event Triggered Model Predictive Controllers-Application to Big Data Cloud Services.* In 55th IEEE International Conference on Decision and Control, 2016. (Cité en pages 20 et 38.)

[Chan 1982] Tony F Chan, Gene Howard Golub et Randall J LeVeque. *Updating formulae and a pairwise algorithm for computing sample variances.* In COMPSTAT 1982 5th Symposium held at Toulouse 1982, pages 30–41. Springer, 1982. (Cité en page 45.)

[Chandola 2009] Varun Chandola, Arindam Banerjee et Vipin Kumar. *Anomaly Detection : A Survey.* ACM Comput. Surv., vol. 41, no. 3, pages 15 :1–15 :58, Juillet 2009. (Cité en pages 19 et 20.)

[Chapelle 2009] Olivier Chapelle, Bernhard Scholkopf et Alexander Zien. *Semi-Supervised Learning (Chapelle, O. et al., Eds. ; 2006)[Book reviews].* IEEE Transactions on Neural Networks, vol. 20, no. 3, pages 542–542, 2009. (Cité en page 24.)

[Chase 2001] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat et Ronald P. Doyle. *Managing Energy and Server Resources in Hosting Centers.* SIGOPS Oper. Syst. Rev., vol. 35, no. 5, pages 103–116, Octobre 2001. (Cité en page 20.)

[Cherkasova 2009] Ludmila Cherkasova, Kivanc Ozonat, Ningfang Mi, Julie Symons et Evgenia Smirni. *Automated Anomaly Detection and Performance Modeling of Enterprise Applications.* ACM Trans. Comput. Syst., vol. 27, no. 3, pages 6 :1–6 :32, Novembre 2009. (Cité en page 26.)

[Cohen 2004] Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons et Jeffrey S. Chase. *Correlating instrumentation data to system states : a building block for automated diagnosis and control.* In Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6, OSDI'04, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association. (Cité en pages 25 et 26.)

[Cooper 2010] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan et Russell Sears. *Benchmarking cloud serving systems with YCSB.* In Proceedings of the 1st ACM symposium on Cloud computing, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM. (Cité en page 79.)

[Davis 2006] Jesse Davis et Mark Goadrich. *The relationship between Precision-Recall and ROC curves.* In Proceedings of the 23rd international conference on Machine learning, pages 233–240. ACM, 2006. (Cité en page 32.)

[Dean 2012] Daniel Joseph Dean, Hiep Nguyen et Xiaohui Gu. *UBL : Unsupervised Behavior Learning for Predicting Performance Anomalies in Virtualized Cloud Systems.* In Proceedings of the 9th International Conference on Au-

tonomic Computing, ICAC '12, pages 191–200, New York, NY, USA, 2012. ACM. (Cité en pages 21, 26 et 32.)

[Denning 1987] D.E. Denning. *An Intrusion-Detection Model.* IEEE Transactions on Software Engineering, vol. SE-13, no. 2, pages 222–232, Feb 1987. (Cité en page 21.)

[Duan 2009] Songyun Duan, Shivnath Babu et Kamesh Munagala. *Fa : A system for automating failure diagnosis.* In 2009 IEEE 25th International Conference on Data Engineering, pages 1012–1023. IEEE, 2009. (Cité en pages 25 et 26.)

[Esfandani 2010] Gholamreza Esfandani et Hassan Abolhassani. *MSDBSCAN : Multi-density Scale-Independent Clustering Algorithm Based on DBSCAN.* In Longbing Cao, Yong Feng et Jiang Zhong, editeurs, Advanced Data Mining and Applications, volume 6440 of *Lecture Notes in Computer Science*, pages 202–213. Springer Berlin Heidelberg, 2010. (Cité en page 27.)

[Eskin 2000] Eleazar Eskin. *Anomaly Detection over Noisy Data using Learned Probability Distributions.* In In Proceedings of the International Conference on Machine Learning, pages 255–262. Morgan Kaufmann, 2000. (Cité en page 22.)

[Ester 1996] Martin Ester, Hans peter Kriegel, Jörg Sander et Xiaowei Xu. *A density-based algorithm for discovering clusters in large spatial databases with noise.* In KDD, pages 226–231. AAAI Press, 1996. (Cité en pages 27 et 28.)

[ETSI Group Speciafication NFV 2013] ETSI Group Speciafication NFV. *Use Cases 001 V1.1.1*, 2013. (Cité en page 93.)

[ETSI 2012] ETSI. *Technical Report 103 125 V1.1.1. CLOUD ; SLAs for Cloud services.* `http://www.etsi.org/deliver/etsi_tr/103100_103199/103125/01.01.01_60/tr_103125v010101p.pdf`, 2012. (Cité en page 42.)

[Farshchi 2015a] M. Farshchi, J. G. Schneider, I. Weber et J. Grundy. *Experience report : Anomaly detection of cloud application operations using log and cloud metric correlation analysis.* In Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on, pages 24–34, Nov 2015. (Cité en pages 21 et 32.)

[Farshchi 2015b] M. Farshchi, J. G. Schneider, I. Weber et J. Grundy. *Experience report : Anomaly detection of cloud application operations using log and cloud metric correlation analysis.* In Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on, pages 24–34, Nov 2015. (Cité en page 25.)

[Foster 2008] I. Foster, Y. Zhao, I. Raicu et S. Lu. *Cloud Computing and Grid Computing 360-Degree Compared.* In 2008 Grid Computing Environments Workshop, pages 1–10, Nov 2008. (Cité en page 9.)

[Fu 2011] Song Fu. *Performance Metric Selection for Autonomic Anomaly Detection on Cloud Computing Systems.* In Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, pages 1–5, 2011. (Cité en page 24.)

[Gander 2013] Matthias Gander, Michael Felderer, Basel Katt, Adrian Tolbaru, Ruth Breu et Alessandro Moschitti. Anomaly detection in the cloud : Detecting security incidents via machine learning, pages 103–116. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. (Cité en page 20.)

[Gardner 1998] Matt W Gardner et SR Dorling. *Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences.* Atmospheric environment, vol. 32, no. 14, pages 2627–2636, 1998. (Cité en page 45.)

[Goldberg 1973] R. P. Goldberg. *Architecture of Virtual Machines.* In Proceedings of the Workshop on Virtual Computer Systems, pages 74–112, New York, NY, USA, 1973. ACM. (Cité en page 11.)

[Goldberg 1974] Robert P. Goldberg. *Survey of Virtual Machine Research.* Computer, vol. 7, no. 9, pages 34–45, Septembre 1974. (Cité en page 11.)

[Gong 2010] Zhenhuan Gong, Xiaohui Gu et John Wilkes. *Press : Predictive elastic resource scaling for cloud systems.* In 2010 International Conference on Network and Service Management, pages 9–16. IEEE, 2010. (Cité en page 22.)

[Grance 2012] Mark L. Badger Tndimothy Grance, Robert Patt-Corner et Jeffrey M. Voas. *Cloud Computing Synopsis and Recommendations.* 2012. (Cité en page 10.)

[Guan 2012a] Qiang Guan, Chi-Chen Chiu, Ziming Zhang et Song Fu. *Efficient and Accurate Anomaly Identification Using Reduced Metric Space in Utility Clouds.* In Networking, Architecture and Storage (NAS), 2012 IEEE 7th International Conference on, pages 207–216, June 2012. (Cité en page 21.)

[Guan 2012b] Qiang Guan, Ziming Zhang et Song Fu. *Ensemble of bayesian predictors and decision trees for proactive failure management in cloud computing systems.* Journal of Communications, vol. 7, no. 1, pages 52–61, 2012. (Cité en pages 28 et 32.)

[Hahsler 2010] Michael Hahsler et Margaret H. Dunham. *rEMM : Extensible Markov Model for Data Stream Clustering in R.* Journal of Statistical Software, vol. 35, no. 5, pages 1–31, 7 2010. (Cité en pages 47 et 50.)

[Hastie 2001] T. Hastie, R. Tibshirani et J.H. Friedman. The elements of statistical learning : Data mining, inference, and prediction. Springer series in statistics. Springer, 2001. (Cité en page 49.)

[Heberlein 1995] L. Heberlein. *Network security monitor (NSM)–final report.* 1995. (Cité en page 21.)

[Hsu 2002] Chih-Wei Hsu et Chih-Jen Lin. *A comparison of methods for multiclass support vector machines.* IEEE Transactions on Neural Networks, vol. 13, no. 2, pages 415–425, Mar 2002. (Cité en page 34.)

[Hsueh 1997] Mei-Chen Hsueh, T. K. Tsai et R. K. Iyer. *Fault injection techniques and tools.* Computer, vol. 30, no. 4, pages 75–82, Apr 1997. (Cité en page 30.)

[Jain 1999] A. K. Jain, M. N. Murty et P. J. Flynn. *Data Clustering : A Review.* ACM Comput. Surv., vol. 31, no. 3, pages 264–323, Septembre 1999. (Cité en page 27.)

[Kundu 2012] Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao et Kaushik Dutta. *Modeling Virtualized Applications Using Machine Learning Techniques.* SIGPLAN Not., vol. 47, no. 7, pages 3–14, Mars 2012. (Cité en page 20.)

[Lakhina 2004] Anukool Lakhina, Mark Crovella et Christophe Diot. *Diagnosing Network-wide Traffic Anomalies.* In Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '04, pages 219–230, New York, NY, USA, 2004. ACM. (Cité en page 26.)

[Laprie 1995] Jean-Claude Laprie, Jean Arlat, Alain Costes et J-P Blanquart. Guide de la sûreté de fonctionnement. Cépaduès-éditions, 1995. (Cité en pages 12 et 29.)

[LeCun 1991] Yann LeCun, Ido Kanter et Sara A Solla. *Second Order Properties of Error Surfaces : Learning Time and Generalization.* In Advances in Neural Information Processing Systems, pages 918–924, 1991. (Cité en page 81.)

[Lee 1999] Wenke Lee, S.J. Stolfo et K.W. Mok. *A data mining framework for building intrusion detection models.* In Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on, pages 120–132, 1999. (Cité en page 21.)

[Lee 2001] Wenke Lee et Dong Xiang. *Information-theoretic measures for anomaly detection.* In Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on, pages 130–143, 2001. (Cité en page 22.)

[Leistner 2009] Christian Leistner, Amir Saffari, Jakob Santner et Horst Bischof. *Semi-supervised random forests.* In 2009 IEEE 12th International Conference on Computer Vision, pages 506–513. IEEE, 2009. (Cité en page 88.)

[Leung 2005] Kingsly Leung et Christopher Leckie. *Unsupervised Anomaly Detection in Network Intrusion Detection Using Clusters.* In Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38, ACSC '05, pages 333–342, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc. (Cité en page 27.)

[Liang 2006] Y. Liang, Y. Zhang, M. Jette, Anand Sivasubramaniam et R. Sahoo. *BlueGene/L Failure Analysis and Prediction Models.* In Dependable Systems and Networks, 2006. DSN 2006. International Conference on, pages 425–434, June 2006. (Cité en page 21.)

[Liang 2007] Yinglung Liang, Yanyong Zhang, Hui Xiong et Ramendra Sahoo. *Failure prediction in ibm bluegene/l event logs.* In Seventh IEEE International Conference on Data Mining (ICDM 2007), pages 583–588. IEEE, 2007. (Cité en page 25.)

[Lone Sang F. 2012] Nicomette V. et Deswarte Y. Lone Sang F. *IronHide : plateforme d'attaques par entrées-sorties.* 2012. (Cité en page 69.)

[Lühr 2009] Sebastian Lühr et Mihai Lazarescu. *Incremental Clustering of Dynamic Data Streams Using Connectivity Based Representative Points.* Data Knowl. Eng., vol. 68, no. 1, pages 1–27, Janvier 2009. (Cité en page 28.)

[Macqueen 1967] J. B. Macqueen. *Some Methods for classification and analysis of multivariate observations.* In Procedings of the Fifth Berkeley Symposium on Math, Statistics, and Probability, volume 1, pages 281–297. University of California Press, 1967. (Cité en pages 27 et 28.)

[Massie 2003] Matthew L. Massie, Brent N. Chun et David E. Culler. *The Ganglia Distributed Monitoring System : Design, Implementation And Experience.* Parallel Computing, vol. 30, page 2004, 2003. (Cité en page 61.)

[Matsunaga 2010] A. Matsunaga et J. A. B. Fortes. *On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications.* In Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, pages 495–504, May 2010. (Cité en page 20.)

[Mazel 2011] Johan Mazel. *Unsupervised network anomaly detection.* PhD thesis, INSA de Toulouse, 2011. (Cité en page 27.)

[Michie 1994] D. Michie, D. J. Spiegelhalter et C.C. Taylor. *Machine Learning, Neural and Statistical Classification*, 1994. (Cité en page 23.)

[Miyazawa 2015] M. Miyazawa, M. Hayashi et R. Stadler. *vNMF : Distributed fault detection using clustering approach for network function virtualization.* In 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 640–645, May 2015. (Cité en pages 27 et 32.)

[Mon 2016] *MongoDB*, 2016. (Cité en page 47.)

[Mukherjee 1994] B. Mukherjee, L.T. Heberlein et K.N. Levitt. *Network intrusion detection.* Network, IEEE, vol. 8, no. 3, pages 26–41, May 1994. (Cité en page 21.)

[Murtagh 1983] Fionn Murtagh. *A survey of recent advances in hierarchical clustering algorithms.* The Computer Journal, vol. 26, no. 4, pages 354–359, 1983. (Cité en page 27.)

[Natella 2016] Roberto Natella, Domenico Cotroneo et Henrique S Madeira. *Assessing Dependability with Software Fault Injection : A Survey.* ACM Computing Surveys (CSUR), vol. 48, no. 3, page 44, 2016. (Cité en page 30.)

[Nguyen 2013] H. Nguyen, Z. Shen, Y. Tan et X. Gu. *FChain : Toward Black-Box Online Fault Localization for Cloud Systems.* In Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on, pages 21–30, July 2013. (Cité en pages 21, 22 et 32.)

[Niwa 2015] T. Niwa, M. Miyazawa, M. Hayashi et R. Stadler. *Universal fault detection for NFV using SOM-based clustering.* In Network Operations and

Management Symposium (APNOMS), 2015 17th Asia-Pacific, pages 315–320, Aug 2015. (Cité en page 27.)

[Pham 2011] Cuong Pham, D. Chen, Z. Kalbarczyk et R.K. Iyer. *CloudVal : A framework for validation of virtualization environment in cloud infrastructure.* In Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on, pages 189–196, 2011. (Cité en page 30.)

[Pop 2016] Daniel Pop. *Machine Learning and Cloud Computing : Survey of Distributed and SaaS Solutions.* CoRR, vol. abs/1603.08767, 2016. (Cité en page 20.)

[Provost 1998] Foster J. Provost, Tom Fawcett et Ron Kohavi. *The Case Against Accuracy Estimation for Comparing Induction Algorithms.* In Proceedings of the Fifteenth International Conference on Machine Learning, ICML '98, pages 445–453, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. (Cité en page 32.)

[Salfner 2007] Felix Salfner et M. Malek. *Using Hidden Semi-Markov Models for Effective Online Failure Prediction.* In Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on, pages 161–174, Oct 2007. (Cité en pages 21 et 22.)

[Salfner 2010] Felix Salfner, Maren Lenk et Miroslaw Malek. *A survey of online failure prediction methods.* ACM Comput. Surv., vol. 42, no. 3, pages 10 :1–10 :42, Mars 2010. (Cité en page 19.)

[Sauvanaud 2015] Carla Sauvanaud, Guthemberg Silvestre, Mohamed Kaâniche et Karama Kanoun. *Data Stream Clustering for Online Anomaly Detection in Cloud Applications.* In 11th European Dependable Computing Conference (EDCC 2015), Paris, France, Septembre 2015. (Cité en pages 73 et 134.)

[Sauvanaud 2016] Carla Sauvanaud, Kahina Lazri, Mohamed Kaâniche et Karama Kanoun. *Anomaly detection and root cause localization in virtual network functions.* In International Symposium on Software Reliability Engineering, Ottawa, Canada, Octobre 2016. (Cité en page 93.)

[Scandariato 2014] R. Scandariato, J. Walden, A. Hovsepyan et W. Joosen. *Predicting Vulnerable Software Components via Text Mining.* IEEE Transactions on Software Engineering, vol. 40, no. 10, pages 993–1006, Oct 2014. (Cité en page 35.)

[Schölkopf 2001] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola et Robert C. Williamson. *Estimating the Support of a High-Dimensional Distribution.* Neural Comput., vol. 13, no. 7, pages 1443–1471, Juillet 2001. (Cité en page 73.)

[Schroeck 2012] Michael Schroeck, Rebecca Shockley, Janet Smart, Dolores Romero-Morales et Peter Tufano. *Analytics : The real-world use of big data.* IBM Institute for Business Value, 2012. (Cité en page 16.)

[Serrano 2016] Damián Serrano, Sara Bouchenak, Yousri Kouki, Frederico Alvares de Oliveira Jr, Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana

Arantes et Pierre Sens. *SLA guarantees for cloud services.* Future Generation Computer Systems, vol. 54, pages 233–246, 2016. (Cité en page 42.)

[Sharma 2013] Abhishek B Sharma, Haifeng Chen, Min Ding, Kenji Yoshihira et Guofei Jiang. *Fault detection and localization in distributed systems using invariant relationships.* In 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 1–8. IEEE, 2013. (Cité en page 22.)

[Shen 2009] Kai Shen, Christopher Stewart, Chuanpeng Li et Xin Li. *Reference-driven Performance Anomaly Identification.* SIGMETRICS Perform. Eval. Rev., vol. 37, no. 1, pages 85–96, Juin 2009. (Cité en page 22.)

[Shon 2007] Taeshik Shon et Jongsub Moon. *A Hybrid Machine Learning Approach to Network Anomaly Detection.* Inf. Sci., vol. 177, no. 18, pages 3799–3821, Septembre 2007. (Cité en page 28.)

[Silvestre 2014] Guthemberg Silvestre, Carla Sauvanaud, Mohamed Kaâniche et Karama Kanoun. *An anomaly detection approach for scale-out storage systems.* In 26th International Symposium on Computer Architecture and High Performance Computing, Paris, France, Octobre 2014. (Cité en pages 45, 46, 68 et 80.)

[Silvestre 2015a] Guthemberg Silvestre, David Buffoni, Karine Pires, Sébastien Monnet et Pierre Sens. *Boosting streaming video delivery with wisereplica.* In Transactions on Large-Scale Data-and Knowledge-Centered Systems XX, pages 34–58. Springer, 2015. (Cité en page 20.)

[Silvestre 2015b] Guthemberg Silvestre, Carla Sauvanaud, Mohamed Kaâniche et Karama Kanoun. *Tejo : A Supervised Anomaly Detection Scheme for NewSQL Databases.* In 7th International Workshop on Software Engineering for Resilient Systems (SERENE 2015), Paris, France, Septembre 2015. (Cité en pages 35, 46, 76, 80 et 81.)

[Simache 2001] Cristina Simache et Mohamed Kaâniche. *Measurement-based availability analysis of Unix systems in a distributed environment.* In Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on, pages 346–355. IEEE, 2001. (Cité en page 21.)

[Simache 2002] Cristina Simache, Mohamed Kaâniche et Ayda Saidane. *Event Log based Dependability Analysis of Windows NT and 2K Systems.* In Dependable Computing, 2002. Proceedings. 2002 Pacific Rim International Symposium on. Citeseer, 2002. (Cité en page 21.)

[Stott 2000] D.T. Stott, B. Floering, D. Burke, Z. Kalbarczpk et R.K. Iyer. *NFTAPE : a framework for assessing dependability in distributed systems with lightweight fault injectors.* In Computer Performance and Dependability Symposium, 2000. IPDS 2000. Proceedings. IEEE International, pages 91–100, 2000. (Cité en page 31.)

[Tan 2010] Yongmin Tan, Xiaohui Gu et Haixun Wang. *Adaptive System Anomaly Prediction for Large-scale Hosting Infrastructures.* In Proceedings of

the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC '10, pages 173–182, New York, NY, USA, 2010. ACM. (Cité en pages 25, 26 et 28.)

[Tan 2012] Yongmin Tan, Hiep Nguyen, Zhiming Shen, Xiaohui Gu, C. Venkatramani et D. Rajan. *PREPARE : Predictive Performance Anomaly Prevention for Virtualized Cloud Systems.* In Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on, pages 285–294, 2012. (Cité en pages 21, 25, 26, 30 et 32.)

[Tu 2009] Li Tu et Yixin Chen. *Stream Data Clustering Based on Grid Density and Attraction.* ACM Trans. Knowl. Discov. Data, vol. 3, no. 3, pages 12 :1–12 :27, Juillet 2009. (Cité en page 28.)

[Van Hulse 2007] Jason Van Hulse, Taghi M Khoshgoftaar et Amri Napolitano. *Experimental perspectives on learning from imbalanced data.* In Proceedings of the 24th international conference on Machine learning, pages 935–942. ACM, 2007. (Cité en page 25.)

[vCloud Suite 2015] vCloud Suite. *VMware vCloud Suite : DATASHEET.* `www.vmware.com/go/vcloud-suite-datasheet`, 2015. (Cité en page 60.)

[Wang 2010] Chengwei Wang, V. Talwar, K. Schwan et P. Ranganathan. *Online detection of utility cloud anomalies using metric distributions.* In 2010 IEEE Network Operations and Management Symposium - NOMS 2010, pages 96–103, April 2010. (Cité en pages 22 et 32.)

[Watanabe 2012] Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi et Y. Matsumoto. *Online failure prediction in cloud datacenters by real-time message pattern learning.* In Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on, pages 504–511, Dec 2012. (Cité en page 21.)

[Wierman 2008] Sonya J. Wierman et Priya Narasimhan. Vajra : Evaluating byzantine-fault-tolerant distributed systems, pages 163–183. John Wiley & Sons, Inc., 2008. (Cité en pages 31 et 69.)

[Williams 2007] A. W. Williams, S. M. Pertet et P. Narasimhan. *Tiresias : Black-Box Failure Prediction in Distributed Systems.* In 2007 IEEE International Parallel and Distributed Processing Symposium, pages 1–8, March 2007. (Cité en page 22.)

[Xiaoyun 2008] Chen Xiaoyun, Min Yufang, Zhao Yan et Wang Ping. *GMDBSCAN : Multi-Density DBSCAN Cluster Based on Grid.* In e-Business Engineering, 2008. ICEBE '08. IEEE International Conference on, pages 780–783, Oct 2008. (Cité en page 27.)

[Xu 2009] Wei Xu, Ling Huang, Armando Fox, David Patterson et Michael I. Jordan. *Detecting Large-scale System Problems by Mining Console Logs.* In Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, SOSP '09, pages 117–132, New York, NY, USA, 2009. ACM. (Cité en pages 21, 26 et 32.)

[Yuan 2014] Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain et Michael Stumm. *Simple Testing Can Prevent Most Critical Failures : An Analysis of Production Failures in Distributed Data-Intensive Systems.* In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 249–265, Broomfield, CO, Octobre 2014. USENIX Association. (Cité en page 63.)

[Zadeh 1965] Lotfi A Zadeh. *Fuzzy sets.* Information and control, vol. 8, no. 3, pages 338–353, 1965. (Cité en page 27.)

[Zhang 1996] Tian Zhang, Raghu Ramakrishnan et Miron Livny. *BIRCH : An Efficient Data Clustering Method for Very Large Databases*, 1996. (Cité en page 27.)

[Zhang 2008] Jiong Zhang, M. Zulkernine et A. Haque. *Random-Forests-Based Network Intrusion Detection Systems.* IEEE Transactions on Systems, Man, and Cybernetics, Part C : Applications and Reviews, vol. 38, no. 5, pages 649–659, Sept 2008. (Cité en pages 25 et 26.)

[Zhang 2013] Yuchao Zhang, Bin Hong, Ming Zhang, Bo Deng et Wangqun Lin. *eCAD : Cloud Anomalies Detection From an Evolutionary View.* In Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on, pages 328–334, Dec 2013. (Cité en pages 21, 27 et 32.)

Résumé

Le cloud computing est un modèle de délivrance à la demande d'un ensemble de ressources informatiques distantes, partagées et configurables. Ces ressources, détenues par un fournisseur de service cloud, sont mutualisées grâce à la virtualisation de serveurs qu'elles composent et sont mises à disposition d'utilisateurs sous forme de services disponibles à la demande. Ces services peuvent être aussi variés que des applications, des plateformes de développement ou bien des infrastructures.

Afin de répondre à leurs engagements de niveau de service auprès des utilisateurs, les fournisseurs de cloud se doivent de prendre en compte des exigences différentes de sûreté de fonctionnement. Assurer ces exigences pour des services différents et pour des utilisateurs aux demandes hétérogènes représente un défi pour les fournisseurs, notamment de part leur engagement de service à la demande. Ce défi est d'autant plus important que les utilisateurs demandent à ce que les services rendus soient au moins aussi sûrs de fonctionnement que ceux d'applications traditionnelles.

Nos travaux traitent particulièrement de la détection d'anomalies dans les services cloud de type SaaS et PaaS. Les différents types d'anomalie qu'il est possible de détecter sont les erreurs, les symptômes préliminaires de violations de service et les violations de service. Nous nous sommes fixé quatre critères principaux pour la détection d'anomalies dans ces services : i) elle doit s'adapter aux changements de charge de travail et reconfiguration de services ; ii) elle doit se faire en ligne, iii) de manière automatique, iv) et avec un effort de configuration minimum en utilisant possiblement la même technique quel que soit le type de service.

Dans nos travaux, nous avons proposé une stratégie de détection qui repose sur le traitement de compteurs de performance et sur des techniques d'apprentissage automatique. La détection utilise les données de performance système collectées en ligne à partir du système d'exploitation hôte ou bien via les hyperviseurs déployés dans le cloud. Concernant le traitement des ces données, nous avons étudié trois types de technique d'apprentissage : supervisé, non supervisé et hybride. Une nouvelle technique de détection reposant sur un algorithme de clustering est de plus proposée. Elle permet de prendre en compte l'évolution de comportement d'un système aussi dynamique qu'un service cloud.

Une plateforme de type cloud a été déployée afin d'évaluer les performances de détection de notre stratégie. Un outil d'injection de faute a également été développé dans le but de cette évaluation ainsi que dans le but de collecter des jeux de données pour l'entrainement des modèles d'apprentissage. L'évaluation a été appliquée à deux cas d'étude : un système de gestion de base de données (MongoDB) et une fonction réseau virtualisée. Les résultats obtenus à partir d'analyses de sensibilité, montrent qu'il est possible d'obtenir de très bonnes performances de détection pour les trois types d'anomalies, tout en donnant les contextes adéquats pour la généralisation de ces résultats.

Abstract

Nowadays, the development of virtualization technologies as well as the development of the Internet contributed to the rise of the cloud computing model. A cloud computing enables the delivery of configurable computing resources while enabling convenient, on-demand network access to these resources. Resources hosted by a provider can be applications, development platforms or infrastructures. Over the past few years, computing systems are characterized by high development speed, parallelism, and the diversity of task to be handled by applications and services.

In order to satisfy their Service Level Agreements (SLA) drawn up with users, cloud providers have to handle stringent dependability demands. Ensuring these demands while delivering various services makes clouds dependability a challenging task, especially because providers need to make their services available on demand. This task is all the more challenging that users expect cloud services to be at least as dependable as traditional computing systems.

In this manuscript, we address the problem of anomaly detection in cloud services. A detection strategy for clouds should rely on several principal criteria. In particular it should adapt to workload changes and reconfigurations, and at the same time require short configurations durations and adapt to several types of services. Also, it should be performed online and automatic. Finally, such a strategy needs to tackle the detection of different types of anomalies namely errors, preliminary symptoms of SLA violation and SLA violations.

We propose a new detection strategy based on system monitoring data. The data is collected online either from the service, or the underlying hypervisor(s) hosting the service. The strategy makes use of machine learning algorithms to classify anomalous behaviors of the service. Three techniques are used, using respectively algorithms with supervised learning, unsupervised learning or using a technique exploiting both types of learning. A new anomaly detection technique is developed based on online clustering, and allowing to handle possible changes in a service behavior.

A cloud platform was deployed so as to evaluate the detection performances of our strategy. Moreover a fault injection tool was developed for the sake of two goals : the collection of service observations with anomalies so as to train detection models, and the evaluation of the strategy in presence of anomalies. The evaluation was applied to two case studies : a database management system and a virtual network function. Sensitivity analyzes show that detection performances of our strategy are high for the three anomaly types. The context for the generalization of the results is also discussed.