Détection et analyse de données aberrantes pour l'adaptation de ressources dans des environnements virtualisés

par

Manel BOULARES

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE DES TECHNOLOGIES DE
L'INFORMATION
M. Sc. A.

MONTRÉAL, LE 11/12/2018

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Mme Nadjia Kara, directrice de mémoire
Département de génie logiciel et de TI à l'École de technologie supérieure

M. Abdelouahed Gherbi, président du jury
Département de génie logiciel et de TI à l'École de technologie supérieure

M. Chamseddine Talhi, membre du jury
Département de génie logiciel et de TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 23/11/2018

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

À mes parents

Je suis particulièrement redevable de votre amour sincère, de votre confiance inconditionnelle et de votre soutien continu pendant mes années d'études. Merci pour tout!

À mon cher mari

Je suis particulièrement reconnaissante pour votre amour, votre compréhension et votre soutien continu. Merci de toujours croire en moi.

À mes frères et ma sœur,

Merci d'être toujours à mes côtés. Aucune dédicace ne saurait exprimer mes sentiments. Je leur souhaite beaucoup de bonheur et de réussite.

Je tiens à remercier tout particulièrement ma directrice de recherche, Mme Nadjia Kara, pour ses conseils, sa disponibilité et son soutien scientifique et moral pendant cette maîtrise.

Un remerciement spécial, à Mlle Souhila Benmakrelouf, Étudiante au Doctorat à l'École de technologie supérieure pour son aide aussi bien technique que morale.

Je tiens en dernier lieu à exprimer ma reconnaissance et ma gratitude à toutes les personnes qui ont contribué de près ou de loin à l'aboutissement de ce travail.

Une pensée à ceux qui nous ont quittés trop tôt

DÉTECTION ET ANALYSE DE DONNÉES ABERRANTES POUR L'ADAPTATION DE RESSOURCES DANS DES ENVIRONNEMENTS VIRTUALISÉS

Manel BOULARES

RÉSUMÉ

L'intégration de l'infonuagique aux technologies de télécommunications a apporté plusieurs avantages aux opérateurs des réseaux et les fournisseurs de services de communication. En effet, l'utilisation de plusieurs nœuds virtuels d'une plateforme de services déployés sur une même machine physique permet de réduire considérablement les coûts et la consommation d'énergie. Avec l'élasticité et l'extensibilité offertes par cette nouvelle technologie, le contrôle des charges de travail dans les systèmes virtualisés est devenu une nécessité afin d'assurer une bonne qualité de service. Dans ce travail de recherche, nous abordons le problème de gestion de la charge de travail dans des infrastructures déployées dans des environnements virtualisés. Notre objectif est de proposer et valider un mécanisme d'adaptation qui assure une gestion dynamique et efficace des ressources dans des environnements virtualisés. L'approche proposée consiste tout d'abord, à tester et analyser les performances d'une plateforme IMS virtualisée que nous avons installée en nous basant sur *OpenIMS Core*. Ensuite, à analyser les données collectées d'IMS afin de détecter les données aberrantes « Outliers potentiels ». Nous proposons également d'effectuer la mise en correspondance entre les métriques de niveau service et celles de niveau ressource à travers l'analyse de leur variation en utilisant la méthode de *Modified z-score*. Cette analyse est basée aussi sur des données extraites d'autres environnements virtualisés tels que Google Cluster et le cloud de l'ÉTS afin d'avoir une solution générique adaptée à plusieurs systèmes virtualisés. En plus, nous effectuons une comparaison entre les deux méthodes de détection des « *Outliers* » à savoir le « *Modified z-score* » et le « *Mahalanobis* ». Notre analyse montre que le « *Mahalanobis* » donne de meilleurs résultats par rapport au « *Modified z-score* ». Les résultats obtenus permettent de repérer les variations importantes pouvant nécessiter une adaptation de ressources dans le système. Ces résultats nous ont permis de concevoir et de développer un algorithme d'adaptation des ressources en nous basant sur la méthode de Mahalanobis.

Mots-clés: Infonuagique, réseau IMS, Virtualisation, SIP, la charge de travail, adaptation des ressources, système IMS, détection d'outliers, la distance de Mahalanobis

DETECTION AND ANALYSIS OF OUTLIERS FOR RESOURCE ADAPTATION IN VIRTUALIZED ENVIRONMENTS

Manel BOULARES

ABSTRACT

The integration of the cloud computing with telecommunications technologies has brought several benefits to network operators and communication service providers. Indeed, the use of several virtual nodes of a service platform deployed on the same physical machine can significantly reduce costs and energy consumption. With the elasticity and scalability offered by this new technology, controlling workloads in virtualized systems has become a necessity to ensure a good quality of service. In this research, we discuss the problem of workload management in virtualized environments. Our goal is to propose and validate an adaptation mechanism that ensures a dynamic and efficient resource management in virtualized environments. The proposed approach consists in the first place of testing and analyzing the performance of a virtualized IMS platform that we installed based on OpenIMS Core. Then, we analyzed the data collected from IMS to detect potential outliers. In addition, we performed the mapping between service-level and resource-level metrics through the analysis of their variation using the modified z-score method. This analysis was also based on data extracted from other virtualized environments such as Google Cluster and the ÉTS in order to have a generic solution adapted to several virtualized systems. In addition, we performed a comparison between the two methods of outliers detection namely modified z-score and mahalanobis. Our analysis showed that mahalanobis gives us better results compared to modified z-score. The results obtained made it possible to identify important variations that may require the adaptation of resources in the system. Finally, we designed and developed a resource adaptation algorithm based on the Mahalanobis method.

Keywords: Cloud computing, IMS network, Virtualization, SIP, workloads, resource adaptation, IMS system, outlier detection, Mahalanobis distance

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CPU	Central Processing Unit (unité centrale de traitement)
EMS	Element Management System
HSS	Home Subscriber Server
IaaS	Infrastructure As A Service
IdO	Internet des Objets
IMS	IP Multimedia Subsystem
IP	Internet Protocol
OSI	Open Systems Interconnection
OS	Operating System
QoS	Quality of Service
SARA	Session-Aware Request Assignment
SCSFs	Call Session Contract Functions
SDN	Software Defined Network
SIP	Session Initiation Protocol
SLA	Service Layer Agreements
SO-E	Service Orchestrator-Execution entity
SO-D	Service Orchestrator-Decision entity
TCP	Transmission Control Protocol
TI	Technologies de l'Information
VN	Virtual Node
VM	Virtual machine
VMM	Virtual machine monitor
v-IMS	Système IMS virtualisé

LISTE DES SYMBOLES ET UNITÉS DE MESURE

APS	Appel par seconde
BP	Bande Passante
GHz	Giga Hertz
Kbps	Kilobit par Seconde
MBps	Mégabit par Seconde
m/s	Message par seconde
S	Seconde
TE	Temps d'exécution
TH	Throughput

INTRODUCTION

Dans un monde où l'échange d'information est devenu très important, plusieurs technologies ont été créées afin de gérer, transférer et stocker cette information. Des nouveaux concepts tels que l'infonuagique et la virtualisation ont permis de mieux optimiser le temps de stockage et de gestion de l'information. L'infonuagique est en train de changer la manière dont les applications informatiques, une des principales sources d'information, sont développées, déployées et gérées puisqu'il offre, à de multiples utilisateurs, des ressources physiques et logicielles sous forme de services. La virtualisation quant à elle, permet d'exploiter les ressources physiques au maximum en fournissant un environnement virtuel aux utilisateurs, offrant ainsi une efficacité, une flexibilité et surtout plus d'économie aux utilisateurs et aux entreprises proposant ces services.

Les fournisseurs d'environnements virtualisés veulent exploiter au maximum les ressources déployées à travers différentes infrastructures réseau afin de garantir un accès libre-service, sur demande et étendu aux ressources et d'assurer des services mesurés et de bonne qualité à leurs clients. La diversité et la demande toujours grandissante pour de nouveaux services offerts à travers des environnements virtualisés poussent ces fournisseurs à adopter et à utiliser des stratégies efficaces d'approvisionnement et de gestion de ressources à travers ces environnements.

Ainsi, le partage efficace des ressources, la réduction des coûts de leurs exploitations et la réduction du temps de leurs gestions sont devenus les nouveaux défis à relever par les fournisseurs des environnements virtualisés.

0.1. Problématique

L'approvisionnement des ressources dans les environnements cloud est une tâche difficile qui peut être compromise en raison de l'indisponibilité ou de l'insuffisance des ressources. La garantie de la qualité de service (QoS) pour les différents services offerts à travers les environnements virtualisés dépend entre autres de ce provisionnement qui doit être équitable

et efficace pour tous les services offerts par les fournisseurs d'environnements virtualisés. De plus, l'adaptation de cet approvisionnement aux exigences de ces services est un problème d'optimisation récurrent. Ainsi, pour garantir une bonne qualité de service et pour minimiser les coûts, il faut avoir des mécanismes efficaces d'adaptation des ressources dans les systèmes virtualisés.

0.2. Objectif de la recherche

L'objectif principal de ce projet de recherche est la définition et la validation d'un algorithme d'adaptation de ressources dans une infrastructure virtualisée tout en garantissant la qualité des services fournis à travers ces environnements. Cet algorithme permet : 1) d'estimer les besoins en ressources des services offerts ; 2) d'adapter les ressources selon ces besoins. Pour estimer les besoins en ressources, l'algorithme établit une correspondance entre des métriques de haut niveau (c.à.d. au niveau service) et des métriques de bas niveau (c.à.d. au niveau ressources physiques). Grâce à cette correspondance, l'algorithme permet de détecter les moments où les ressources doivent être adaptées pour différents services. Il permet d'évaluer le sens de variations des ressources (augmentation ou réduction des ressources) ainsi que la quantité de ressources à ajouter ou supprimer selon cette variation.

0.3. Méthodologie de travail

Pour atteindre l'objectif établi de notre recherche, nous proposons de diviser le travail en trois parties. La première partie consiste en l'analyse des systèmes à étudier. Différentes données (ex. CPU, Bande passante) sont collectées de divers systèmes virtualisés (ex. système IMS, application web). Ces relevés de données sont utilisés pour analyser les systèmes virtualisés étudiés afin de mieux comprendre leurs comportements. Pour analyser le comportement de ces systèmes en termes de ressources consommées (ex. CPU, Bande passante), nous proposons d'adapter deux méthodes de détection de valeurs aberrantes pour chaque type de ressource analysée. La deuxième partie consiste en l'adaptation et l'analyse des approches *Modified Z-score* (Iglewicz et Hoaglin, 1993) et *Mahalanobis* (Mahalanobis, 1936). Cette étape est très importante, car elle nous permet d'identifier la méthode de détection de données aberrantes la plus efficace et la plus adaptée aux comportements des systèmes virtualisés étudiés. La

dernière partie consiste à implémenter un algorithme d'adaptation des ressources pour des systèmes virtualisés. La solution se base sur la méthode de détection d'outliers la plus fiable afin d'adapter les ressources au moment approprié pour assurer une bonne qualité de service et de minimiser la perte de ressources. Le mécanisme conçu est suffisamment générique puisqu'il permet d'adapter les ressources pour différents systèmes virtualisés.

0.4. Organisation du rapport

Ce mémoire est divisé en quatre chapitres. Le premier présente une revue de l'état de l'art où nous présentons certains concepts clés tel que la virtualisation et l'infonuagique. Nous analyserons aussi les mécanismes de contrôle et de surveillance de trafic dans de tels environnements ainsi que certains concepts fondamentaux de l'analyse de données. Dans le deuxième chapitre, nous décrirons la première méthode de détection de données aberrantes appelée le « Modified Z-score ». Dans le troisième chapitre, nous présentons la deuxième méthode de détection de données aberrantes appelée « le Mahalanobis » que nous comparerons avec le Modified Z-score. Dans le quatrième chapitre, nous décrirons l'algorithme d'adaptation de ressources basé sur la méthode « Mahalanobis ». La conclusion et les travaux futurs sont présentés à la fin de ce mémoire.

CHAPITRE 1

REVUE DE LA LITTÉRATURE

Introduction

Ce chapitre présente une revue de la littérature reliée aux mécanismes de contrôle et de gestion des ressources dans des environnements virtualisés. Nous commençons tout d'abord par la définition de certains concepts liés à notre projet de recherche. Ces concepts présentent les notions de la virtualisation, de l'infonuagique et des architectures utilisées dans les réseaux de communication virtualisés. Ensuite, nous présentons un survol sur les travaux liés à la gestion et à l'adaptation des ressources dans de tels systèmes. Enfin, ce chapitre se termine par une description des méthodes utilisées pour la détection des valeurs aberrantes.

1.1 Définitions et contexte de base

L'évolution technologique a permis à des nouveaux concepts de prouver leurs importances dans le domaine de l'informatique comme la virtualisation et l'infonuagique. Ce dernier est considéré comme l'une des innovations technologiques les plus importantes dans ce domaine. Il est en train de changer la manière dont les applications informatiques sont déployées, développées et gérées puisqu'il offre des ressources physiques et logicielles sous forme de service à des multiples utilisateurs. La virtualisation quant à elle, permet d'exploiter les ressources physiques en fournissant un environnement virtuel aux utilisateurs offrant ainsi une efficacité, une flexibilité et surtout plus d'économie aux utilisateurs et aux entreprises proposant ce service.

1.1.1 La virtualisation

Le travail de recherche de (Calarco et Casoni, 2013) analyse les techniques de la virtualisation, les opportunités qu'elle offre, les problèmes et les limitations de cette technologie ainsi que son efficacité dans les réseaux hétérogènes. Un réseau hétérogène est un réseau multi fournisseurs où tous les composants matériels ou logiciels proviennent de différents fournisseurs. L'implémentation des réseaux hétérogènes est souvent une tâche difficile surtout

lorsque différentes technologies sont nécessaires, il est difficile de prédire si une infrastructure peut accomplir la performance opérationnelle attendue. Dans cet article, les auteurs ont utilisé des techniques de virtualisation basée sur les conteneurs et un outil de simulation de réseau pour montrer qu'un certain nombre de réseaux virtuels distincts peut être exécuté en même temps. Leurs buts étaient de concevoir un outil nommé NetBoxIT pour soutenir la création et l'interconnexion de plusieurs réseaux virtuels sur une seule plate-forme physique multi-cœur. Ils ont défini la virtualisation comme étant l'abstraction des fonctionnalités de composants physiques.

Vu que les ressources physiques ne peuvent pas être illimitées, la virtualisation est la meilleure solution permettant d'économiser et de fournir un service efficace et rapide. Cette technologie supporte l'augmentation continue des demandes en ressources tout en permettant à plusieurs systèmes d'exploitation invités virtuels (machines virtuelles ou GuestOSs) de partager ces ressources et de les exploiter au maximum. Un système virtualisé utilise une couche logicielle dédiée, appelée hyperviseur, moniteur de machine virtuelle ou VMM, pour planifier et gérer les ressources partagées entre plusieurs applications. Par contre, dans un système non virtualisé un seul système d'exploitation est utilisé pour gérer les ressources matérielles entre les applications. (Liao, Leung et Chen, 2015)

1.1.2 L'infonuagique

Pendant des années, les opérateurs des réseaux et les fournisseurs de services de communication ont souffert du manque des ressources où ils ont dû faire face à l'évolution des applications en ajoutant plus de serveurs pour pouvoir répondre aux besoins des consommateurs. De ce fait, les entreprises disposaient d'un très grand nombre de serveurs dont les ressources (ex. processeurs et la mémoire) étaient utilisées à un faible pourcentage de leurs capacités. Le *cloud computing* a été proposé comme solution à ce problème de gestion inefficace des ressources. Il permet la consolidation des serveurs, l'économie d'énergie et il facilite la rapidité de déploiement avec un coût plus faible. Puisque le cloud utilise les technologies de virtualisation, il hérite des avantages liés à cette technologie. Le cloud et la virtualisation sont deux concepts bien différents. Cependant, les deux permettent une gestion

plus facile, rapide et plus efficace des ressources. Cette évolution des ressources et des technologies a permis de revoir la façon avec laquelle les opérateurs de réseaux et les fournisseurs de services gèrent ces ressources. (Ahad, Chan et Santos, 2015) Le principal défi auquel ces entreprises doivent faire face est de garder le même niveau de qualité de service pour les utilisateurs.

Les auteurs de l'article de (Wei et al., 2013) ont montré qu'IMS *« IP multimedia subsystem »* est un exemple de système favorable pour l'intégration du cloud computing puisqu'il fournit : un contrôle de signalisation ouvert et standardisé, un contrôle de QoS[1] négociable, un service réutilisable et des interfaces normalisées et uniformes de cloud computing. En plus, IMS utilise le *Session Initiation Protocol* SIP pour la signalisation et la gestion de session. SIP est un protocole de signalisation de la couche application du modèle OSI permettant l'établissement et la fin des sessions multimédias telle que la vidéo/audio conférence, téléphonie, notification d'événements et la messagerie instantanée.

En tant que plate-forme de service uniforme, IMS offre une gestion et un contrôle unifié pour tous les services. Les capacités de services communs d'IMS existant, tels que la gestion de présence et de groupe, pourraient être exposées et réutilisées par les services de cloud via des interfaces standardisées. De la même façon, certains services de cloud de base, comme Amazon EC2 et S3, peuvent également être réutilisés pour créer des services de cloud computing à valeur ajoutée. Les utilisateurs finaux ont besoin d'un seul authentifiant pour l'accès à tous les services autorisés IMS, y compris divers services de cloud computing. (Montazerolghaem et al., 2016)

Ainsi, nous souhaitons adapter les ressources de différents systèmes virtualisés tels que IMS d'une façon équitable et efficace. Pour cela, dans notre travail de recherche nous allons nous

[1] Quality of Service : est la capacité à fournir un service conforme à des exigences en matière de temps de réponse, de bande passante, etc.

baser sur trois systèmes virtualisés distincts tels qu'IMS, le cloud de l'ÉTS et les grappes de Google (Google Clusters 2011).

1.2 Mécanisme de contrôle et de surveillance de trafic dans les environnements cloud

L'environnement cloud nécessite un mécanisme de contrôle et de surveillance de la charge de trafic performant et efficace pour faire face aux fluctuations de cette dernière. Dans ce qui suit, nous proposant d'analyser les mécanismes de contrôle et de surveillance des flux dans des environnements non virtualisés et virtualisés.

1.2.1 Mécanismes de contrôle et de surveillance des flux dans des environnements non virtualisés

Avec l'évolution de la technologie et de réseau Internet, les demandes et les exigences des utilisateurs de réseau augmentent et évoluent continuellement. Face à cette évolution, les fournisseurs de service Internet cherchent à toujours garantir la qualité de service. Notre travail de recherche s'est basé, entre autres sur l'étude des flux générés dans le sous-système multimédia basé sur *IP* (*IMS)*. IMS utilise le SIP comme protocole de contrôle de sessions et d'appels. Le SIP « Session Initiation Protocol » est un protocole très utilisé dans le domaine de télécommunications permettant d'établir et de gérer les sessions dans les réseaux IP. La configuration des serveurs pour une application basée sur SIP n'est pas facile. Le débordement ou la surcharge des serveurs peut nuire à la performance de l'application. D'où l'idée de créer des méthodes de prévention et de gestion de surcharge afin de maintenir la performance des serveurs SIP.

Dans l'article (Sun et al., 2007), les chercheurs ont créé un système *« Front End »* de gestion des flux appelé *« FEFM : Front End SIP Flow Management System »* pour améliorer la performance et la protection des serveurs SIP *« Back End »*. Le système proposé permet de :

1) Mettre les sessions encours dans des files d'attente et limite le nombre des sessions simultanées dans les serveurs, ce qui réduit le risque de surcharge.

2) Estimer le temps de réponse des requêtes permettant l'établissement des sessions INVITE et affecte des priorités aux demandes qui pourraient répondre à l'exigence *SLA « Service Layer Agreement »* lors de la planification des messages.
3) Refuser les demandes de sessions établit lorsque la charge est trop élevée ce qui protège les performances des sessions acceptées.
4) Supprimer les retransmissions permettant d'alléger la charge de serveur.

En 2009,(Jing et al., 2009) se sont inspiré par le travail de Sun et al et ont proposé une autre solution de gestion des flux pour le système IMS. Ils considèrent le problème de protection contre la surcharge des serveurs en estimant en avance la consommation des ressources en termes de CPU pour chaque requête envoyée dans le réseau et en supprimant les retransmissions en double des files d'attente. Aussi, ils prévoient le temps de réponse de chaque session établie et la priorisent selon les critères de QoS. Cette solution est destinée aux environnements de télécommunications non virtualisés. Elle ne prend en compte qu'un type de requête à la fois.

Dans l'article (Jiang et al., 2009) , les auteurs ont utilisé le SIP *« Session Initiation Protocol »* comme un protocole de signalisation permettant de contrôler et de supporter plusieurs types de médias afin de tester le trafic dans le réseau. C'est un protocole basé sur les transactions permettant de débuter et de terminer des sessions média. L'état d'une session commence par la transaction INVITE et se termine par le BYE, chaque transaction SIP crée un état de session valable pour la durée de la session. L'idée de créer des algorithmes de contrôle et de surveillance vient du besoin des entreprises fournissant ce service de gérer plusieurs centaines d'utilisateurs voire des milliers. Les demandes et les requêtes des utilisateurs dans le réseau augmentent au fur et à mesure. Elles doivent être contrôlées et reparties d'une façon optimale pour supporter un déploiement à large échelle. La recherche effectuée présente un mécanisme central basé sur des algorithmes d'orchestration des demandes permettant de répartir et d'équilibrer la charge de trafic. Ce mécanisme peut être utilisé aussi avec d'autres systèmes où il est avantageux d'inclure un outil d'équilibrage de charge pour maintenir des sessions dans lesquelles les demandes correspondant à la même session sont envoyées au même serveur.

SARA *« Session-Aware Request Assignment »* est un processus permettant l'attribution des sessions aux serveurs disponibles dans l'infrastructure dédiée et garantit l'acheminement des demandes de mêmes sessions aux mêmes serveurs. L'utilisation de cette méthode permet de rendre la performance du mécanisme de répartition de charge plus efficace où il peut utiliser les informations issues de SARA afin d'améliorer le temps de réponse et le débit.

Ces mêmes auteurs ont publié un autre travail relié à la répartition des charges de trafic. Dans cet article, Jiang et al (Jiang et al., 2012) ont présenté différents algorithmes utilisés dans le système d'équilibrage de charges. Ils décrivent une nouvelle approche utilisée pour la répartition de charge. L'approche consiste à identifier les paquets SIP par leur Call-ID et de l'utiliser comme une clé pour l'identification des appels. Ce système de répartition de charge achemine la première demande d'un appel au serveur approprié et maintient également les correspondances entre les appels et les serveurs en utilisant deux tables de hachage qui sont indexées par l'ID de l'appel. De cette façon, quand une nouvelle transaction correspondant à l'appel est reçue, elle sera redirigée vers le bon serveur. La première table de hachage a pour rôle de conserver les informations concernant l'état des appels et les transactions en cours de traitement. Quant à la deuxième table, elle est utilisée pour le routage des paquets en double pour les demandes terminées.

Dans le tableau 1.1, nous résumons les travaux de recherche effectués dans le domaine de la gestion des flux dans les réseaux de télécommunication tout en indiquant les contributions apportées et les limitations identifiées pour chaque approche.

Tableau 1.1 Gestion des flux dans les environnements cloud

	Solution proposée	**Résultats**	**Limitations**
(Sun et al., 2007)	Améliorer la performance et la protection des serveurs SIP	1. Protection de surcharge 2. Rejet des retransmissions 3. Gestionnaire de temps de réponse	1. Solution non générique, spécifique à IMS 2. Ne gère pas l'équilibrage de charge 3. Environnement non virtualisé

Tableau 1.1 Gestion des flux dans les environnements cloud (Suite)

	Solution proposée	Résultats	Limitations
		4. Ordonnancement avec deux files d'attente	
(Jing et al., 2009)	Gérer des flux pour le système IMS	1. Protection contre la surcharge des serveurs 2. Estimation en avance la consommation de CPU pour chaque requête 3. Suppression des retransmissions en double des files d'attente.	1. Solution non générique, spécifique à IMS 2. IMS non virtualisé 3. Traite un seul type de requête
(Jiang et al., 2009)	Gérer des sessions, SARA « Session-Aware Request Assignment » : Acheminer les demandes de mêmes sessions aux mêmes serveurs	1. Répartition efficace de charge 2. Amélioration de temps de réponse et de débit	1. Solution non générique, spécifique à IMS 2. IMS non virtualisé
(Jiang et al., 2012)	Implémenter un mécanisme d'équilibrage de charge dans les serveurs SIP	Amélioration de débit et de temps de réponse	1. Solution non générique, spécifique à IMS 2. Métrique bas niveau non considéré (ex. CPU) 3. IMS non virtualisé

1.2.2 Mécanismes de contrôle et de surveillance des flux dans des environnements virtualisés

Pour tester les performances des environnements virtualisés, différents mécanismes de contrôle et de surveillance des flux ont été proposés.

Par exemple, la plateforme IMS a été virtualisée pour gérer les flux de trafic des composants de l'internet des objets IdO. Le but de travail de Been et al (Been et al., 2015) était d'avoir un outil permettant la gestion et la surveillance du trafic dans cette plateforme.
L'infrastructure proposée se compose d'un administrateur d'un système IMS virtualisé (v-IMS) et d'un contrôleur SDN. Le premier gère le v-IMS. Il permet de créer, de maintenir et de supprimer des v-IMS. Il permet aussi de surveiller l'état des v-IMS. Le deuxième composant de cette infrastructure qui est le contrôleur SDN (basé sur le réseau défini par le logiciel), il effectue le classement et le routage du trafic de IdO via un autre module qui gère le trafic IdO. En outre, le contrôleur SDN commande et gère le trafic de signal envoyé par les terminaux mobiles afin de diminuer la charge de v-IMS où il achemine le trafic au v-IMS le moins chargé en utilisant une table de routage. Dans cet article, les chercheurs ont montré que l'utilisation de SDN basée sur un réseau virtualisé permet une gestion efficace des ressources réseau. Cette infrastructure leur permet d'atteindre leur but ultime qui est d'optimiser l'utilisation des ressources tout en garantissant un coût faible.

De plus, plusieurs travaux de recherche ont traité le problème d'approvisionnement de ressources dans le cloud. Ces travaux peuvent être divisés en deux catégories : approches statiques et approches dynamiques.

Une approche statique a été introduite par (Bedhiaf, Cherkaoui et Pujolle, 2009). Cette approche permet la comparaison entre 2 environnements virtualisés. Les critères sur lesquelles se base cette comparaison est la charge de trafic de signalisation et le temps de réponse. Pour modéliser la demande de service virtuel, ils ont utilisé un modèle basé sur les priorités pour l'allocation des ressources. La solution proposée est spécifique à IMS et permet l'amélioration des délais de l'enregistrement et de l'établissement de sessions, mais elle ne considère pas les critères de qualité de service. Il faut noter que dans cette étude, l'allocation des ressources est statique. Il serait donc possible d'améliorer encore les performances en proposant une allocation dynamique et d'évaluer le comportement des ressources avec des métriques de bas niveau (utilisation du *CPU*, mémoire, bande passante).

Les auteurs dans [(Melo et al., 2012), (Lu et al., 2013)] ont proposé aussi des approches statiques. Avec ces approches, les fournisseurs de service n'ont pas la possibilité d'ajuster les ressources pour leurs nœuds virtuels déployés. Ils considèrent les contraintes de capacité et de performance de l'infrastructure physique lors du mappage des VN sans considérer les besoins en QoS (ex, CPU, bande passante, mémoire, etc.) de chaque nœud virtuel. Leurs solutions supposent que l'allocation des ressources s'effectue implicitement une fois le meilleur chemin pour un VN est identifié en allouant une portion fixe de chaque nœud et de chaque lien et permettent de trouver la meilleure correspondance d'un nœud virtuel sur un réseau physique en identifiant le meilleur chemin pour un VN. Cette correspondance est effectuée en termes d'utilisation de CPU au niveau des nœuds et de la bande passante au niveau du lien. Cette proposition génère des problèmes lors du partage des charges où il peut y avoir une sous-utilisation des ressources physiques suite à la pré-allocation des ressources physiques. Ainsi, dans ce cas, un nœud virtuel peut utiliser le maximum de la capacité du CPU du nœud et toute la bande passante d'un lien.

Pour résoudre ces problèmes susmentionnés, les approches dynamiques viennent remédier à ce défaut en offrant la possibilité d'adapter les ressources disponibles dans chaque nœud virtuel. Les approches développées se sont concentrées surtout sur l'allocation d'une seule métrique telle que la bande passante. En effet, plusieurs travaux de recherche se sont concentrés sur l'étude de la consommation de la bande passante par les machines virtuelles dans les réseaux. Tan et al (Tan et al., 2014) ont proposé un algorithme d'ajustement de la bande passante d'une manière adaptative. Ils considèrent que prévenir et ajuster la consommation de la bande passante selon le besoin de nœuds virtuels dans le réseau est meilleur que de réagir après avoir eu des baisses performances. De ce fait, la migration des machines virtuelles n'était pas une solution envisageable pour ces chercheurs parce qu'ils préfèrent avoir une solution logicielle de prévention et d'adaptation que de faire la migration des machines virtuelles et changer l'infrastructure du réseau virtuel.

Wenzhi et al. (Wenzhi et al., 2012) quant à eux, ont traité le problème de la répartition de la bande passante entre les différentes machines virtuelles d'une manière équilibrée dans un

environnement de virtualisation de réseaux. La solution proposée repose sur la pression de la charge interne du lien afin d'allouer la capacité nécessaire selon le besoin en termes de bande passante entre les différents nœuds virtuels. Les auteurs ont conçu un algorithme d'ordonnancement des paquets afin d'équilibrer la pression sur chaque lien. Cet algorithme permet d'allouer la bande passante en répartissant la capacité du lien entre les différents nœuds.

Le travail de Carella et al (Carella et al., 2014) a permis de prouver la faisabilité du déploiement d'*IMS* comme un service (*IMS* as a Service : *IMSaaS*) sur les infrastructures infonuagiques des réseaux mobiles. Il décrit un mécanisme d'orchestration des services dans les réseaux virtualisés. L'orchestration des services proposée se compose d'une entité de décision (*SO-D*) et d'une entité d'exécution (*SO-E*) qui s'occupe du déploiement du service et de la gestion du temps d'exécution à travers la communication avec le contrôleur de l'infonuagique et le système de gestion des éléments (*EMS*). Ce travail présente une architecture de gestion permettant de simplifier le déploiement et l'orchestration d'un tel service virtuel dans une infrastructure cloud. La solution implémentée permet de supporter une charge de 50 appels téléphoniques par seconde. Ainsi, les résultats des expérimentations ont monté que le test s'arrêtait autour de 50 appels par seconde (APS) à cause de la limitation d'utilisation de la mémoire du *S-CSCF* (2 *GB* de *RAM*). Ils ont considéré que les niveaux atteints d'appels par seconde satisfont des opérateurs de téléphonie fixe ou mobile de taille moyenne ayant besoin de charges moyennes. Mais pour des périodes de pointes (charges élevées) ou des opérateurs de téléphonie de plus grande taille, *vIMS* doit être étendu (*scaled*). Plus d'expérimentations sur la plateforme avec différents scénarios (varier les ressources et les charges de travail) et une analyse des performances plus détaillée pourront renforcer les conclusions annoncées.

Les auteurs Wang, Tay et Golubchik (Wang, Tay et Golubchik, 2012) proposent un mécanisme d'allocation de ressources basé sur l'analyse de l'interaction entre le comportement de l'utilisateur et les performances du réseau. Ils considèrent l'équilibre du trafic comme un équilibre entre une entrée contrôlée par les utilisateurs et une sortie contrôlée par le réseau (par exemple, capacité de liaison, contrôle de la congestion, etc.). Le nombre de connexions actives est contrôlé par les utilisateurs et l'état du réseau affecte la rapidité avec laquelle une connexion

peut être établie. Les auteurs de ce travail mesurent les probabilités des requêtes web abandonnées par l'utilisateur lorsqu'il n'est pas satisfait du service, le taux d'arrivée des paquets et le débit de la connexion TCP. À partir de ces valeurs, ils estiment la bande passante nécessaire pour chaque utilisateur. Il peut, par la suite, ajuster dynamiquement sa demande en fonction de ses besoins.

Une autre approche a été proposée par Gong et Wilkes. La solution implémentée est appelée Press « PRedictive Elastic reSource Scaling ». (Gong, Gu et Wilkes, 2010) Cette solution est basée sur des algorithmes de traitement de signal et d'apprentissage statistique pour prédire en temps réels les besoins en ressources pour différentes applications. Les auteurs ont montré que leur approche permet d'allouer la quantité de ressources nécessaires aux applications, d'éviter la dégradation de service et de minimiser le gaspillage des ressources. PRESS estime en permanence les besoins en ressources des applications et prédit les ressources dans un proche avenir. Il utilise la détection des cycles afin de prédire la quantité de ressources nécessaires. Les modèles de prédiction de ressources sont mis à jour à plusieurs reprises lorsqu'il détecte un changement au niveau de ces cycles afin d'adapter les ressources aux nouveaux besoins. Cependant, PRESS donne une priorité plus élevée pour éviter la sous-estimation que pour éviter la surestimation.

La plupart des solutions existantes ne tiennent pas compte de la correspondance et le lien entre les différentes ressources dans les réseaux virtualisés. Ces solutions cherchent principalement à offrir un meilleur partage des ressources de l'infrastructure physique sans tenir compte de l'équité entre les ressources bas niveau et haut niveau. Le tableau 1.2 résume les principales solutions proposées dans la littérature. Nous nous sommes inspirés dans notre travail de recherche des travaux de (Wang, Tay et Golubchik, 2012) et (Gong, Gu et Wilkes, 2010), pour développer une approche de prédiction de ressources dans les environnements virtualisés. Cette approche utilise des modèles statistiques pour la prédiction de la charge de trafic dans ces environnements. Elle se base sur des mesures relevées aussi bien au bas niveau (ex. CPU, bande passante) que haut niveau (ex., throughput). Ainsi, dans ce travail de recherche nous nous intéressons à analyser les données collectées (ex. ressources consommées, throughput

généré) et à adapter les ressources (ex. CPU) selon l'allure de ces données. Cette analyse consiste à relever les données aberrantes qui indiqueraient un besoin d'adapter les ressources d'un système.

Tableau 1.2 Tableau récapitulatif des solutions proposées pour gérer le trafic dans des environnements virtualisés

	Solution proposée	**Résultats**	**Limitations**
(Been et al., 2015)	Gestion de trafic IoT pour les systèmes IMS virtualisé	1. Distribution adéquate des ressources 2. Utilisation d'un système de classification de trafic	1. Solution non générique 2. QoS non considérée
(Wei et al., 2013)	Approvisionnement dynamique des ressources	1. Utilisation efficace des ressources 2. Mécanisme de contrôle et de notification	1. Solution non générique 2. QoS non considérée
(Lu et al., 2013)	Virtualisation du cœur du réseau d'IMS	1. Allocation dynamique des ressources 2. Migration des MV en temps réel 3. Équilibrage de charge 4. Rétablissement après catastrophe	1. Solution non générique 2. Un seuil prédéfini pour l'allocation des ressources 3. QoS non considérée, réaction lente
(Bedhiaf, Cherkaoui et Pujolle, 2009)	Plusieurs scénarios de virtualisation des noeuds d'IMS (CSCFs, HSS).	Amélioration des délais de l'enregistrement et l'établissement de sessions	1. Allocation statique des ressources 2. SLA, QoS non considérés 3. Solution non générique (spécifique à IMS)

Tableau 1.2 Tableau récapitulatif des solutions proposées pour gérer le trafic dans des environnements virtualisés (Suite)

	Solution proposée	Résultats	Limitations
(Carella et al., 2014)	IMS est déployé comme un service dans les infrastructures infonuagiques des réseaux mobiles.	Orchestration des services	1. Charge limitée à 50 appels par seconde 2. Solution non générique, implémentée pour gérer le flux IMS
(Wenzhi et al., 2012)	Solution pour la répartition de la bande passante entre les différents nœuds dans un environnement virtualisé	1. Répartition de la bande passante d'une manière équitable 2. Respecte les différents QoS	1. Ne considère que la bande passante 2. Solution non générique
(Tan et al., 2014)	Méthode dynamique de répartition de la bande passante pour différents liens virtuels	1. Ajustement adaptatif de la bande passante 2. Utilisation des files d'attente avec un faible coût	1. Ne considère que la bande passante 2. Solution non générique

1.2.3 Concepts fondamentaux de l'analyse de données

Afin de comprendre le comportement d'un système virtualisé, une analyse de son état au niveau des ressources (ex. CPU, bande passante) et au niveau des services (ex. délai de bout-en-bout, throughput) est nécessaire. La méthode choisie pour faire cette analyse est la détection des valeurs aberrantes. Cette technique nous permet de savoir à quel moment le système rencontre des difficultés en termes d'approvisionnement de ressources ou bien en termes de dégradation de service.

1.2.3.1 La détection des valeurs aberrantes

Il existe plusieurs interprétations d'une valeur aberrante, appelée aussi «outlier». Selon (Barnett et Lewis, 1994), une valeur aberrante est « *une observation (ou sous-ensemble d'observations) qui semble être incompatible avec le reste de cet ensemble de données* ». Cependant, la détection de valeurs aberrantes nécessite une analyse et une évaluation concrète afin de s'assurer que la valeur identifiée représente bien un outlier. Par exemple, dans les environnements de télécommunications, l'arrivée en rafale de trafic voix cause la surcharge des serveurs et les performances du système de communication baissent. Plusieurs raisons peuvent engendrer une rafale de trafic : les périodes occupées durant les heures de pointe dans le cas d'un centre d'appel, les attaques de dénis de service ou encore des demandes d'enregistrement après le rétablissement du courant électrique dans un système de communication. La consommation des ressources bas niveau (ex. CPU, Memoire) peut aussi être dûe à des mises à jour dans le système. Par exemple, des mises à jour peuvent entrainer une augmentation de la consommation du CPU. Le système identifie cette observation comme un outlier, mais qui peut être en réalité un faux outlier, le système ne rencontre pas une baisse de performance et la qualité de service ne se dégrade pas. (Sinha, 1979)

Plusieurs méthodes sont utilisées afin de détecter les outliers pour ensuite les supprimer de l'ensemble des données [(Inc et Ratner, 2012) , (Maciá-Pérez et al., 2015), (Maheshwari et Singh, 2016)]. Dans ce contexte, la suppression de bruit causé par les outliers permet d'avoir une distribution plus stable. Cependant cette technique est loin d'être parfaite parce qu'on doit analyser les causes menant à ces valeurs aberrantes et ensuite identifier s'il y a lieu d'éventuelles dégradations des performances. D'où l'importance d'utiliser les méthodes de détection des outliers qui établissent une correspondance entre les données de bas niveau (ex. CPU, Mémoire) et haut niveau (ex. délai et throughput).

Plusieurs méthodes de détection des outliers existent[(Cao et al., 2017), (Lorido-Botran et al., 2017), (Freeman, 1995)]. La nature de données ainsi que l'objectif de l'étude effectuée ont un impact sur le choix de la méthode à utiliser. Les méthodes de détection des valeurs aberrantes

peuvent être classées en deux grandes catégories. La première est l'analyse uni-variée et la deuxième c'est l'analyse multivariée.

L'approche uni-variée analyse une variable à la fois. Son objectif est de décrire la variable étudiée. Tandis que l'approche bi-variée ou multivariée implique l'analyse de deux variables ou plus afin d'identifier toute association ou relation possible.
Les méthodes les plus utilisées et les plus populaires pour les données uni-variées normalement distribuées sont la méthode Z-score, la méthode z-score modifiée et le test de Grubbs. Cependant, il est recommandé d'utiliser les distances comme mesure de calcul de similitude pour le type de données multivariées. Les distances permettent d'exposer l'ensemble de points corrélés dans un ensemble de données et de calculer la distance de chaque point par rapport à cet ensemble. Les deux méthodes les plus utilisées pour ce type de données sont le Mahalanobis Distance et le Cook's Distance. (Inc et Ratner, 2012) Dans ce projet, nous allons utiliser l'approche Mahalanobis pour détecter les valeurs aberrantes. Cette méthode nous permet d'analyser la corrélation entre les données étudiées. Elle consiste à calculer la distance de Mahalanobis en utilisant la matrice de covariance et le tableau moyen. Nous allons aussi comparer l'approche Mahalanobis avec l'approche Modified Z-score.

Presque toutes les méthodes uni-variées et multivariées sont basées sur l'hypothèse de données normalement distribuées, une condition nécessaire qui doit être satisfaite. Si l'hypothèse de normalité n'est pas satisfaite, la décision «il y a une valeur aberrante » peut être due à la non-normalité des données plutôt qu'à la présence d'une valeur aberrante. (Inc et Ratner, 2012)

1.2.3.2 Comparaison entre les méthodes de détection des outliers

Le tableau 1.3 montre la différence entre les deux types approches uni-variées et multivariées:

Tableau 1.3 Méthodes de détection d'outliers uni-variée et multivariée

Données uni-variées	**Données multivariées**
Implique une seule variable	Implique deux variables et plus
Ne se base pas sur la corrélation	Se base sur la corrélation et la relation entre les données
L'objectif principal de cette analyse est la description	L'objectif principal de cette analyse est l'explication
• Analyse une seule variable à la fois • La moyenne, le mode, la médiane, l'étendue, la variance, le maximum, le minimum, les quartiles et l'écart-type.	• Analyse deux variables simultanément • Corrélation • Comparaison, relations, causes, explication
Méthodes d'affichage • Tables de distribution de fréquence • Diagrammes à barres • Histogrammes • Polygones de fréquence • Diagrammes à secteurs	Méthodes d'affichage • Diagramme de dispersion, • Analyse de corrélation • Analyse par grappes • Analyse de correspondance • Analyse factorielle • Analyse multidimensionnelle • Analyse de régression multiple et partielle • Analyse de redondance

Bien que la description soit l'objectif principal des données de type uni-varié, les méthodes statistiques multivariées peuvent être utilisées à des fins descriptives selon l'objectif de l'étude effectuée. La plus grande similitude entre les techniques statistiques uni-variées et multivariées est qu'elles sont toutes les deux importantes pour la compréhension et l'analyse de données statistiques détaillées. L'analyse uni-variée agit comme un précurseur de l'analyse multivariée et une analyse préliminaire est parfois nécessaire pour comprendre cette dernière. (Yuen et Mu, 2012) Dans le cadre de ce projet, nous proposons d'appliquer ces méthodes pour la détection des valeurs aberrantes dans des systèmes virtualisés.

Conclusion

Dans ce chapitre nous avons défini quelques concepts importants à savoir la virtualisation et le cloud. Ensuite, nous avons présenté quelques mécanismes de contrôle, de gestion et de

surveillance de flux dans les réseaux virtualisés et non-virtualisés. Finalement, nous avons introduit la notion de détection d'outliers et les méthodes que nous allons utiliser afin d'adapter les ressources et d'éviter la dégradation des services.

Le chapitre suivant décrit en détail la notion de détection des valeurs aberrantes et la méthode de Modified Z-score utilisée.

CHAPITRE 2

ANALYSE DES PERFORMANCES DE L'ALGORITHME DE DÉTECTION D'OUTLIERS BASÉ SUR LA MÉTHODE MODIFIED Z-SCORE

Introduction

Dans le cadre de la gestion des ressources dans des environnements cloud, nous sommes amenés à étudier le comportement de différents systèmes virtualisés. L'étude se base sur le besoin en termes de ressources physiques (ex., CPU, Bande passante) consommées par ces systèmes. Pour cela, nous proposons d'analyser les données collectées à partir de ces systèmes et d'évaluer la quantité de ressource consommée ainsi que la variation de cette consommation. Cette analyse vise à détecter les valeurs aberrantes, appelées aussi « *outliers* », qui représentent des variations inhabituelles de consommation de ressources. Ces outliers nous permettront de détecter les moments appropriés pour adapter des ressources (ex. ajouter ou supprimer des ressources) des systèmes virtualisés. Dans ce chapitre, nous allons analyser une première approche de détection d'outliers. Cette approche de détection d'outliers basée sur la méthode Modified Z-score pour des systèmes virtualisés a été proposée par Mlle Souhila Benmakrelouf, étudiante au doctorat à l'ÉTS. Nous proposons de tester et d'analyser les performances de cette approche pour différents systèmes virtualisés (ex. IMS, application Web) et types de ressources (ex., CPU et bande passante).

2.1 Analyse de l'approche de détection d'outliers basée sur la méthode Modified Z-score

Dans cette sous-section, nous allons présenter l'approche de détection d'outliers basée sur la méthode Modified Z-score. Avant de décrire l'approche et d'analyser ses performances, nous allons définir l'ensemble des termes utilisés pour cette approche.

2.1.1 Définitions

Une valeur aberrante (en anglais, *outlier*) est une observation qui sort de l'ordinaire et qui semble différente et incompatible des autres observations dans un ensemble de données. Pour

détecter ces outliers, nous pouvons utiliser plusieurs méthodes permettant d'identifier ces différentes variations.(Sinha, 1979)

La détection des outliers : est reconnue comme une méthode très importante permettant l'exploration de données. Il s'agit de la découverte de phénomènes anormaux qui peuvent exister dans un ensemble de données. Autrement dit, des valeurs de données qui s'écartent considérablement des autres valeurs de données.(Cao et al., 2017)

Le modified z-score : est l'une des méthodes qui permet d'identifier des outliers. Elle a été proposée par Iglewicz et Hoaglin. Elle est basée sur la déviation standard et le z-score pour identifier les valeurs aberrantes. (Iglewicz B. et Hoaglin D., 1993)

2.1.2 La méthode Modified Z-score

Dans cette section, nous décrivons l'approche de détection de données aberrantes basée sur la méthode Modified Z-score. Cette dernière est utilisée afin de comprendre le comportement des différents systèmes virtualisés que nous allons étudier. Le tableau 2.1 représente les différents symboles utilisés dans l'application de cette méthode.

Tableau 2.1 Tableau des symboles utilisés

Symbole	Définition
Z_i	La valeur de Z-score d'une observation i
X_i / x_i	La valeur observée
$\overline{X}$	La moyenne de l'échantillon étudié
S_d	La déviation standard
n	Le nombre d'observations
$\frac{(n-1)}{\sqrt{n}}$	Le score Z maximal
M_i	La valeur de modified z-score de la valeur observée i
MAD	Median Absolute Deviation qui est la médiane de $\{\|x_i - \bar{x}\|\}$
$\bar{x}$	La médiane de l'ensemble de données
S	Le seuil de modified z-score

Le Z-score :

Le Z-score d'une observation est défini comme suit :

$$Z_i = \frac{X_i - \bar{X}}{s_d} \quad (2.1)$$

Avec $\bar{X}$ est la moyenne de l'échantillon et s_d est la déviation standard (c'est l'écart-type des observations).

Bien que cette procédure soit simple pour détecter les valeurs aberrantes, $\bar{X}$ et s_d sont fortement affectés par les valeurs aberrantes et sont sujettes à un effet de masque (Iglewicz et Hoaglin, 1993; Barnett et Lewis, 1994), un effet qui se produit lorsque les observations avec les valeurs les plus extrêmes ne sont pas pris en compte dans l'analyse ce qui empêche de prédire et de déclarer les observations ultérieures comme étant aberrantes. Ce phénomène est particulièrement présent dans les échantillons de petite taille en raison du fait que la moyenne et l'écart-type sont fortement affectés par les valeurs aberrantes et que le score Z maximal est au plus égal à $\frac{(n-1)}{\sqrt{n}}$.

Le modified Z-score (M-zscore) :

Cette méthode est une modification du z-score conventionnelle, où elle prend comme paramètre le MAD (Median Absolute Deviation) qui est la médiane de $\{|x_i - \bar{x}|\}$ et $\bar{x}$ est la médiane de l'ensemble de données. Elle est définie par l'équation (2.2) :

$$M_i = \frac{0.6745\ (x_i - \bar{x})}{MAD} \quad (2.2)$$

Où E (MAD) = 0,675 σ lorsque X est normalement distribué. Iglewicz et Hoaglin (1993) ont suggéré que les observations soient marquées comme valeurs aberrantes lorsque | Mi | > 3,5 grâce à la simulation basée sur des observations pseudo-normales pour différentes tailles d'échantillon. Le Z-score modifié est efficace pour les données qui suivent approximativement une loi normale de la même manière que le Z-score. (Kannan, Manoj et Arumugam, 2015)

Le seuil sert à distinguer et à identifier les outliers de l'ensemble de données étudiées. Si le score en valeur absolue d'une observation dépasse le 3,5 alors elle est considérée comme étant une valeur aberrante.

2.1.3 Description de l'approche proposée

Dans cette partie, nous allons présenter l'approche proposée pour détecter les outliers. L'algorithme 2.1 permet de calculer la variation des valeurs moyennes des données observées et d'extraire les paires d'outliers pour les deux métriques étudiées (ex. CPU et throughput).

Algorithme 2.1 Détection d'outliers avec le Modified Z-score

Input : Metric1, Metric2, Datapath
Output : O_metric1, O_metric2, O_pairs, O_pairs_dist

1. **Function OD_Mzscore (CPU,BP,datapath)**
2. dataset1= [Metric1 Metric2]
3. size = length(dataset1)
4. TimeSlot = [1:1:size]'
5. % Calcul des valeurs moyennes des métriques étudiées sur n données observées
6. [meanM1,i] = average(dataset1(:,1), 3)
7. [meanM2,j] = average(dataset1(:,2), 3)
8. % Calcul de la variation des métriques sur n données observées
9. [varM1,index3] = variation(meanM1,i)
10. [varM2,index1] = variation(meanM2,j);
11. % Calcul de la valeur maximale pour chaque métrique sur n données observées
12. [maxM1,i] = maximum(dataset1(:,1), 3)
13. [maxM2,j] = maximum(dataset1(:,2), 3)
14. meandata = [meanM2 meanM1 maxM2 maxM1]
15. vardata = [varM2 varM1]
16. %Extraire les paires d'outliers des 2 métriques étudiées
17. outliers(i,dataset1, meandata(1:j,:), vardata,'ETS Data, SMTP',datapath)
18. **Return** O_metric1, O_metric2, O_pairs, O_pairs_dist
19. **End**

L'algorithme 2.2 permet d'extraire les outliers pour chaque métrique étudiée et de faire la correspondance entre les résultats des deux métriques.

Algorithme 2.2 Extraction des outliers détectés

```
Input : I, dataset1, meandata, vardata
Output : O_metric1, O_metric2, O_pairs, O_pairs_dist
1.  Function outliers(i,dataset1, meandata, vardata)
2.  %initialisation
3.   O_metric1 = ('No outliers have been identified!');
4.   O_metric2 = ('No outliers have been identified!');
5.   O_pairs = ('No outliers have been identified!');
6.   O_pairs_dist = ('No outliers have been identified!');
7.   size =i;
8.   threshold = 3.5;  % Modified Z-Score threshold
9.   TimeSlot = [1:1:size]';
10. % Détection des outliers avec la variation moyenne des données avec le modified
    z-score
11.  [mzscore1 maxmz1 outlier1 outlier_num1] = mzscore(vardata(1:size-1,1),
    x_date(1:size-1,1), threshold, 0);
12.  [mzscore2, maxmz2 outlier2 outlier_num2] = mzscore(vardata(1:size-1,2),
    x_date(1:size-1,1), threshold, 0);
13. % Faire la correspondance entre les deux métriques étudiées
14.  s=1;
15.  if isnumeric(outlier_num1) && isnumeric(outlier_num2)
16.  for i=1:length(outlier_num1(:,1))
17.     var_M1(i)=mzscore1(outlier_num1(i,1),1);
18.     max_M1(i)=meandata(outlier_num1(i,1),3);
19.          for j=1:length(outlier_num2(:,1))
20.             var_M2(j)=mzscore2(outlier_num2(j,1),1);
21.             max_M2(j)=meandata(outlier_num2(j,1),3);
22.             if (outlier_num1(i,1)==outlier_num2(j,1))
23.                outlier_pairs(s,:) = [outlier_num1(i,1) s];
24.                 s=s+1;
25.                 O_pairs=outlier_pairs(:,1);
26.            End
27.        End
28.  End
29. %extraire les paires d'outliers avec leurs scores
30. if isnumeric(O_metric1)
31.         O_metric1=[outlier_num1(:,1), var_M1', max_M1']
32.         O_metric2=[outlier_num2(:,1), var_M2', max_M2']
33. Else
34.        error(' No matching have been identified.')
35. End
36. %chercher le maximum de valeur observée pour chaque outlier
37. if isnumeric(outlier2)
38.     maxvarB=meandata(1:size-1,4)
```

Algorithme 2.2 Extraction des outliers détectés (Suite)

```
39.    for v=1:length(outlier_num2(:,1))
40.       maxBP=num2str(maxvarB(outlier_num2(v,1),1))
41.       maxBP(v,:)=maxBP
42.    End
43. End
44. if isnumeric(outlier1)
45.    z=1
46.    while z<=length(outlier_num1(:,1))
47. %     Calculer la distance entre deux outliers
48.     if z<=length(outlier_pairs(:,1))
49.        if z==1
50.          dist=num2str(outlier_pairs(1,1))
51.            v= num2str(outlier_pairs(1,1))
52.        elseif (z<=length(outlier_pairs(:,1)))
53.         dist(z)=outlier_pairs(z,1)-outlier_pairs(z-1,1)
54.          for k=z:length(dist)
55.              v = [v,'+',num2str(dist(z))]
56.          End
57.        End
58.     End
59.    z=z+1
60.    End
61.    if isnumeric(dist)
62.    O_pairs_dist=dist'
63.    End
64. Else
65.   error('No outliers have been identified.')
66. End
67. Return O_metric1, O_metric2, O_pairs, O_pairs_dist
68. End
```

L'algorithme 2.3 permet de calculer les scores des métriques étudiées en suivant la méthode de Modified z-score et de vérifier la normalité des données.

Algorithme 2.3 Calculer le modified Z-score

```
Input : x, x_date, thresh, dist
Output : mzscore maxmz outlier outlier_num
   1.  Function mzscore(x, x_date, thresh, dist)
   2.  % Vérifier le nombre des entrées
   3.   if (nargin < 2) || (nargin > 4)
   4.      error('Requires two to four input arguments.')
```

Algorithme 2.3 Calculer le modified Z-score (Suite)

```
5.  end
6.  % Définir les valeurs par défaut
7.   if nargin == 2
8.      thresh = 3.5
9.      dist = 0
10.  elseif nargin == 3
11.    dist = 0
12.  end
13.  % Transformation normale10: if dist == 1
14.      x = log(x)
15.   End
16.  % Vérifier la normalité des données
17.   if ~isnumeric(x) || ~isreal(x) || ~iscellstr(x_date)
18.        error('Input x must be a numeric array, x must be positive for log-normality,
 and x_date must be a string table.')
19.    end
20.  %Calculer la function de modified z-score
21.   [n, c] = size(x)
22.   mad = median(abs((x-repmat(median(x),n,1))))
23.   mzscore = 0.6745*(x-repmat(median(x),n,1))./repmat(mad,n,1)
24.   [i,j] = find(abs(mzscore) > thresh)
25.   maxmz = (n-1)/sqrt(n)
26.  %Traiter le cas où aucun outlier n'a été détecté
27.   if ~isempty(i)
28.       outlier =[]
29.       outlier_num = [i j]
30.  else
31.       outlier = ('No outliers have been identified!')
32.       outlier_num = ('No outliers have been identified!')
33.   end
34.   Return mzscore maxmz outlier outlier_num
35.  End
```

L'algorithme 2.4 permet de calculer la valeur de variation des valeurs moyennes pour 3 données observées. L'approche proposée par Mlle Souhaila donne des meilleurs résultats en calculant la variation pour 3 données consécutives.

Algorithme 2.4 Calculer la variation des valeurs observées

```
Input : dataVector,size
Output : variationData,Index
1. Function variation( dataVector, size)
2. % calculer la variation de chaque n observations dans l'ensemble de données
3. variationData = zeros(size-1, 1); %initialisation
4.   for i= 2:1:size
5.         variationData(i-1,1)= (dataVector(i,1) - dataVector(i-1,1));
6.   End
7.  index = i-1
8.  Return variationData, index
9. End
```

2.2 Environnement de test

Dans cette partie nous détaillerons notre environnement de test. Pour pouvoir créer une solution générique, nous avons choisi d'analyser les données de 3 systèmes différents à savoir : le sous-système multimédia basé sur IP (IMS) virtualisé, différentes applications de l'environnement cloud de L'ÉTS (ex. HTTTP, SMTP) et les différentes applications de Google.

Le tableau 2.2 décrit la configuration de chaque système virtualisé et les données utilisées dans le but d'étudier le comportement des 3 systèmes. Pour chaque système, nous avons choisi des métriques de bas niveau de son infrastructure infonuagique telles que le CPU et la bande passante et une métrique de haut niveau (niveau service) telle que le *throughput*.

Tableau 2.2 Environnement de test

IMS	ETS	Google
Description		
Système étudié : le sous-système multimédia basé sur IP (IMS) virtualisé. **Métriques :** - CPU : représenté en %. - Throughput (TH) : mesuré en message par seconde (m/s).	**Système étudié** : les applications du cloud l'ÉTS. **Métriques :** - CPU : est représenté en %. - Bande passante (BP) : est en KB/s.	**Système étudié** : les grappes de serveurs de Google : données 2011. **Métriques :** - CPU : est représenté en %. - Temps d'exécution d'une tâche (procédure exécutée

Tableau 2.2 Environnement de test (Suite)

IMS	ETS	Google
Description		
		par le même travail (Job) sur une machine virtuelle), en s.
Scénarios de déploiement utilisés		
Scénario Numéro 1 : Taille de données = 93 valeurs observées pour les deux métriques CPU et Throughput.	Scénario Numéro 5 : SMTP (Simple Mail Transfert Protocol) Taille de données = 333 valeurs observées pour les deux métriques CPU et bande passante.	Scénario Numéro 8 : Tâche Numéro 8 Taille de données = 146 valeurs observées pour les deux métriques CPU et temps d'exécution.
Scénario Numéro 2 : Taille de données = 178 valeurs observées pour les deux métriques CPU et Throughput.	Scénario Numéro 6 : WEB Taille de données = 333 valeurs observées pour les deux métriques CPU et bande passante.	Scénario Numéro 9 : Tâche Numéro 26 Taille de données = 147 valeurs observées pour les deux métriques CPU et temps d'exécution.
Scénario Numéro 3 : Taille de données = 142 valeurs observées pour les deux métriques CPU et Throughput.	Scénario Numéro 7: SQL (Structured Query Language) Taille de données = 333 valeurs observées pour les deux métriques CPU et bande passante.	Scénario Numéro 10 : Tâche Numéro 42 Taille de données = 147 valeurs observées pour les deux métriques CPU et temps d'exécution.
Scénario Numéro 4 : Taille de données = 78 valeurs observées pour les deux métriques CPU et Throughput.	-	Scénario Numéro 11 : Tâche Numéro 57 Taille de données = 147 valeurs observées pour les deux métriques CPU et temps d'exécution.
-	-	Scénario Numéro 12 : Tâche Numéro 85 Taille de données = 147 valeurs observées pour les deux métriques CPU et temps d'exécution.

2.3 Hypothèses et scénarios concrets

2.3.1 Hypothèses

Dans notre projet, nous avons analysé la consommation de ressource de différents systèmes virtualisés. Les données ont été collectées aussi bien au niveau plan de contrôle (ex., consommation de ressources pour l'échange de la signalisation SIP d'un système IMS) qu'au niveau plan de données (ex., consommation de ressources par le trafic HTTP, SMTP, SQL). Ces données peuvent être divisées en deux grandes catégories : la première catégorie est de type TI, c'est les données issues de google et des applications cloud de l'ÉTS (ex., données web, SMTP, SQL), et la deuxième catégorie est de type Télécommunications (ex., données collectées à partir du système virtualisé d'IMS).

Les hypothèses de travail sur lesquelles nous nous sommes basées sont les suivantes :

1) Pour Google, les configurations de l'infrastructure restent cachées/anonymes. On suppose que ces données sont collectées pour chaque machine virtuelle. Elles ne représentent pas la concaténation de plusieurs systèmes virtualisés.
2) On suppose que les données utilisées sont des données réelles qui n'ont pas été modifiées et que ces valeurs collectées reflètent bien l'état du système et des ressources consommées.
3) Les outils de surveillance et de collecte de données sont fiables et que les tâches exécutées par ces outils n'altèrent pas les valeurs relevées.

2.3.2 Définitions

Dans la partie qui suit, nous allons présenter quelques scénarios concrets de l'application de la méthode de Modified Z-score. Nous allons analyser les performances de cette dernière appliquée pour différents systèmes virtualisés en modifiant à la fois la taille de l'ensemble des données étudiées et le seuil.

Pour les différents ensembles de données collectées, nous avons choisi d'étudier l'impact de la taille de données étudiées sur les résultats obtenus avec le Modified Z-score. Nous avons

choisi de débuter avec un nombre de données bien défini que nous appelons échantillon. Nous appliquons le Modified Z-score à cet ensemble de données et nous prélevons les résultats obtenus. Ensuite, nous ajoutons à chaque fois le même nombre de valeurs de l'échantillon choisi et nous analysons les résultats. (ex., nous débuterons l'analyse avec un ensemble de 6 valeurs ensuite nous ajoutons 6 valeurs et nous appliquons le Modified Z-score à ce nouvel ensemble de données et nous comparons les résultats du premier sous-ensemble de données de 6 valeurs au deuxième sous-ensemble de données de 12 valeurs).

Avant de décrire ces scénarios, nous allons définir quelques notions :

Échantillon : ensemble de valeurs à étudier extrait des données collectées.

Sous-ensemble : un ensemble des données extrait d'un échantillon.

Numéro de sous-ensemble : identifiant du sous-ensemble étudié.

Seuil : La valeur de seuil S utilisé pour l'analyse servant à identifier les outliers. Nous allons étudier l'impact du changement de la valeur du seuil sur les outliers détectés. Nous prenons comme valeurs 3, 3.5 et 4. Les données ayant un score supérieur à ce seuil est considéré comme un outlier.

Nombre d'outliers/position : Le nombre de données aberrantes détectées et leurs positions dans le temps.

Valeurs de Mzscore : La valeur de Modified Z-score pour chaque outlier détecté.

[Min,Max] mzscore : la valeur minimale et maximale de Modified Z-score de l'échantillon analysé.

Moyenne de Mzscore : la moyenne des scores de l'échantillon.

Pour les deux systèmes virtualisés IMS et ÉTS, nous définissons trois scénarios pour lesquels différents seuils seront testés. Les mêmes remarques et les mêmes explications se répètent pour les grappes de Google. Le tableau 2.3 représente les différents scénarios des tests effectués. Nous avons choisi des échantillons multiples de3 puisque la valeur de modified z-score prend la variation des 3 valeurs observées pour le calculer du score.

Tableau 2.3 tableau représentant les scénarios des tests effectués

Type d'application	**Les tests**		
IMS : Scénario de déploiement numéro 2	Test 1 Échantillon **12 valeurs** **Seuil [3.5, 3, 4]**	Test 2 Échantillon **18 valeurs** **Seuil [3.5, 3, 4]**	Test 3 Échantillon **6 valeurs** **Seuil [3.5, 3, 4]**
ÉTS : Scénario de déploiement numéro 5			

2.3.3 Scénarios pour l'application IMS

Afin d'étudier les performances de l'approche Modified Z-score, nous avons analysé l'impact de la taille des données et du seuil sur les résultats obtenus. Ainsi, nous avons analysé la variation du nombre d'outliers en fonction de la taille de données et de la valeur du seuil. Dans ce qui suit, nous avons choisi l'ensemble des données collectées pour la métrique CPU que nous allons analyser pour différents scénarios. Prenons par exemple le test 1 du scénario 2 :

❖ **Scénario2 -test 1**

Dans ce scénario de test, nous avons collecté 14 sous-ensembles de données. Nous avons analysé les résultats en variant le seuil comme suit :

- **Seuil = 3.5 :** Pour ce seuil, on trouve 3 sous-ensembles de données où nous pouvons remarquer l'apparition existence des nouveaux outliers. Ces résultats sont représentés au niveau du tableau III.1 (voir annexe III). Le premier outlier apparait dans le sous-ensemble 4 à la position 14. La valeur de Modified Z-score peut être négative ou positive, ce signe nous permet de savoir le sens de variation des données. Si la variation est positive alors il y a une augmentation dans la métrique étudiée par contre si la variation est négative, on peut déduire qu'il y a une diminution de la métrique observée par rapport à l'ensemble de données.

- **Seuil = 3 :** Les résultats obtenus avec un seuil égal à 3 sont représentés dans le tableau III.2 (voir annexe III). Nous remarquons quelques changements au niveau des outliers détectés comparativement aux résultats obtenus avec un seuil 3.5. En effet, il y a eu des

changements au niveau des outliers détectés dans 5 sous-ensembles plutôt que 3 sous-ensembles identifiés pour un seuil de 3.5. Un nouveau outlier est apparu dans le sous-ensemble numéro 12 à la position 46. Cet outlier a comme score -3.1896. On peut conclure qu'en diminuant le seuil, nous obtenons un nombre plus élevé d'outliers, ceci est dû au fait que plusieurs valeurs ayant un score entre 3,5 et 3 vont être considéré comme outlier. Cependant, nous remarquons la disparition de l'outlier dans le sous-ensemble 13. En augmentant la taille des données étudiées, on peut conclure que les outliers avec un petit score sont affectés par le reste de données d'où la disparition de l'outlier à la position 46. Prenons le même exemple de l'outlier 46, il est apparu dans le sous-ensemble 12 avec un score égal à |-3.1896|. Le score de cet outlier a diminué et a atteint la valeur de |-2.7376|. Cette diminution peut être expliquée par l'augmentation du nombre des valeurs observées. Les scores calculés par le Modified Z-score augmentent et diminuent selon la variation et la valeur des données observées.

- **Seuil = 4 :** En augmentant le seuil nous remarquons que les outliers détectés avec le seuil 3,5 restent inchangés pour les trois premiers sous-ensembles. Pour le sous-ensemble 13, l'outlier 14 n'est pas affichée dans le Tableau III.3 (voir annexe III), car sa valeur Mzscore est égale à 3.7098. Elle est donc inférieure au seuil choisi. La valeur de Modified z-score est affectée par la valeur de la variation de l'ensemble de données étudiées, s'il y a une grande variation en termes de données nous remarquons une augmentation dans le score calculé et vice versa. On remarque qu'on augmentant la valeur du seuil, nous obtenons moins d'outliers. Plusieurs valeurs ayant un score entre 3,5 et 4 ne sont pas considérées comme outliers.

Les tests 2 et 3 du scénario 2 ainsi que les tests 1,2 et 3 du scénario 5 sont décrits dans l'annexe III. Dans ce qui suit, nous présentons les principales observations faites à partir de cette analyse :

- Les valeurs des scores des outliers détectés par la méthode de Modified Z-score restent les mêmes en changeant le seuil. Pour savoir si ces outliers sont vrais ou pas, nous

allons les comparer dans la sous-section 2.5 aux données réelles de chaque métrique et voir leurs variations.

- Généralement, en augmentant la taille de l'échantillon, la valeur de Modified Z-score peut augmenter ou diminuer ceci peut être expliqué par le fait que la méthode utilisée se base sur la déviation absolue de la médiane et la médiane de l'ensemble de données. Donc le score est impacté par les valeurs voisines. Ces dernières peuvent soit réduire la variation ou l'augmenter.
- En diminuant le seuil, on peut obtenir plus d'outliers et en l'augmentant on obtient moins d'outliers. La valeur de seuil a été fixée par (Iglewicz B. et Hoaglin D., 1993) à 3.5.
- Les outliers détectés à une position donnée et qui disparaissent en augmentant le nombre des valeurs de l'échantillon étudié sont impactés par la valeur des données observées.

Ces tests nous ont permis de valider et de comprendre le fonctionnement de la méthode de Modified Z-score. Dans ce qui suit, nous allons appliquer cette méthode en prenant en compte les différents scénarios mentionnés à la section 2.3 avec un seuil de 3.5.

2.4 Validation de l'approche de détection d'outliers basée sur la méthode Modified Z-score

Dans cette section, nous allons mesurer la fiabilité de l'approche de détection d'outliers basée sur la méthode Modified Z-score avec un seuil de 3.5. Nous allons comparer les résultats obtenus avec la méthode de Modified Z-score et les données réelles collectées des différents systèmes virtualisés. La fiabilité F est mesurée comme suit :

$$F = \frac{\text{Nombre des vrais outliers}}{\text{Nombre totale des outliers détectés}} \tag{2.3}$$

Un vrai outlier est une valeur qui nous indique un comportement anormal ou inhabituel du système virtualisé étudié. Il permet de savoir qu'il y a une sous-utilisation ou une surutilisation des ressources tandis qu'un faux outliers est une fausse alerte où le système peut indiquer des

grandes variations de données sans avoir un impact sur les ressources (ex. lors des mises à jour d'un système).

2.4.1 Analyse des données d'IMS

2.4.1.1 Scénario 1

Dans ce scénario de déploiement, aucun outlier n'a pu être détecté par l'approche Modified Z-score (scénario décrit à la section 2.3). Les figures 2.1 et 2.2 montrent respectivement l'allure de la valeur de M-zscore et les valeurs observées des deux métriques étudiées à savoir CPU et throughput (TH).

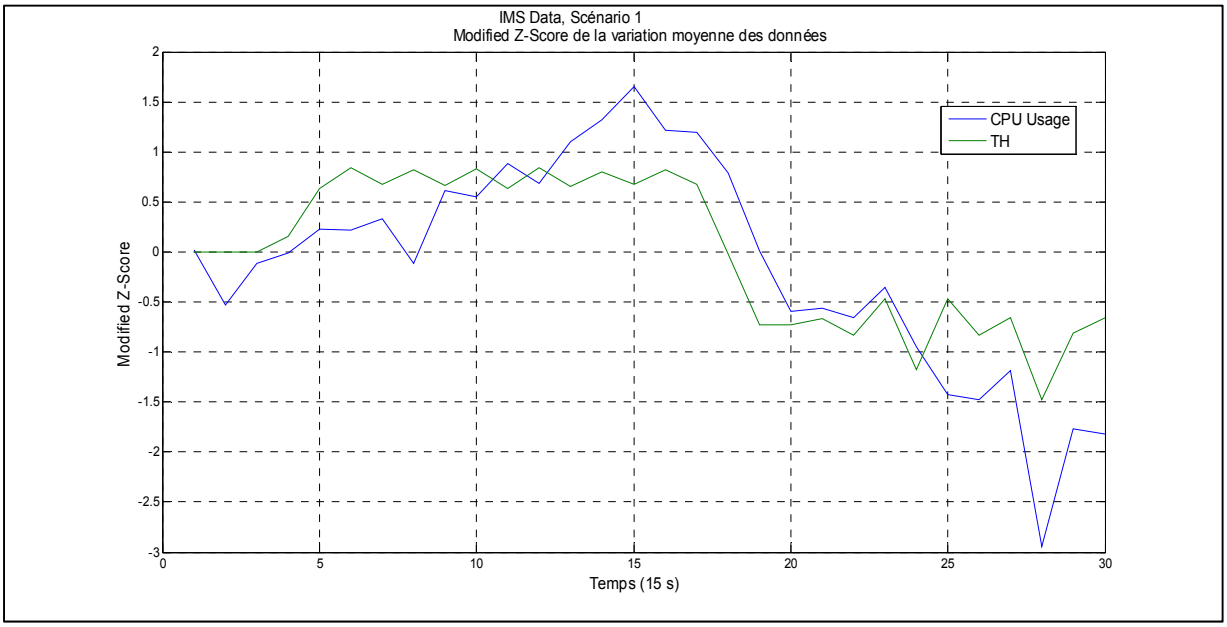

Figure 2.1 Scénario 1 - Valeur de Modified z-score CPU et TH

Le Modified Z-score ne dépasse pas le 1,5 pour l'ensemble des valeurs observées des deux métriques. Ceci peut être expliqué par l'allure des données observées pour le CPU et le TH (voir figure 2.2). Nous remarquons qu'il n'y a pas une grande variation pour les deux métriques ni de valeurs aberrantes. Nous remarquons que les deux courbes de la figure 2.2 n'ont pas la même allure. En effet, une métrique augmente l'autre diminue et vice versa.

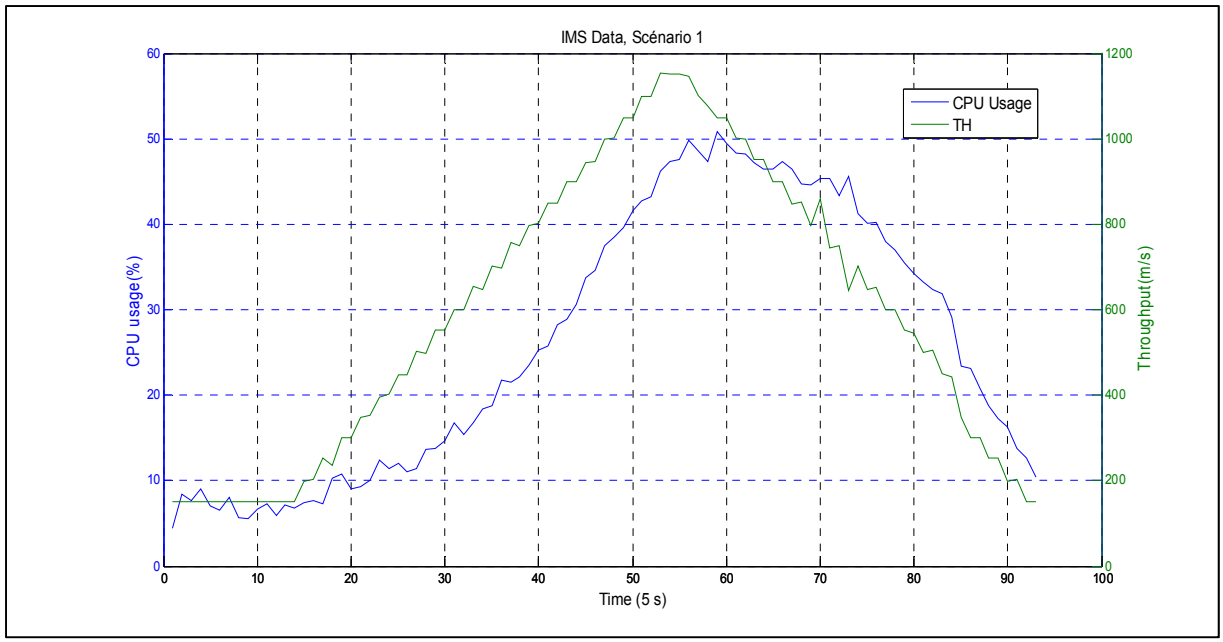

Figure 2.2 Scénario 1 - Variation des données réelles CPU et TH

Le CPU dans ce scénario ne dépasse pas le 50% et les deux métriques ont un comportement différent ceci peut être expliqué par le flux des messages envoyés dans le réseau pour la métrique TH. Dans la figure 2.2, aucun outlier n'a été représenté parce que pour ce scénario de déploiement nous n'avons pas détecté des valeurs qui sortent de l'ordinaire. La valeur de modified z-score n'a pas dépassé 3,5 donc nous ne pouvons pas mesurer la fiabilité de la méthode de Modified z-score.

2.4.1.2 Scénario 2

Dans ce scénario, nous remarquons la présence de plusieurs outliers pour les deux métriques CPU et TH (voir figure 2.3). Nous nous intéressons aux outliers détectés pour les deux métriques au même moment.

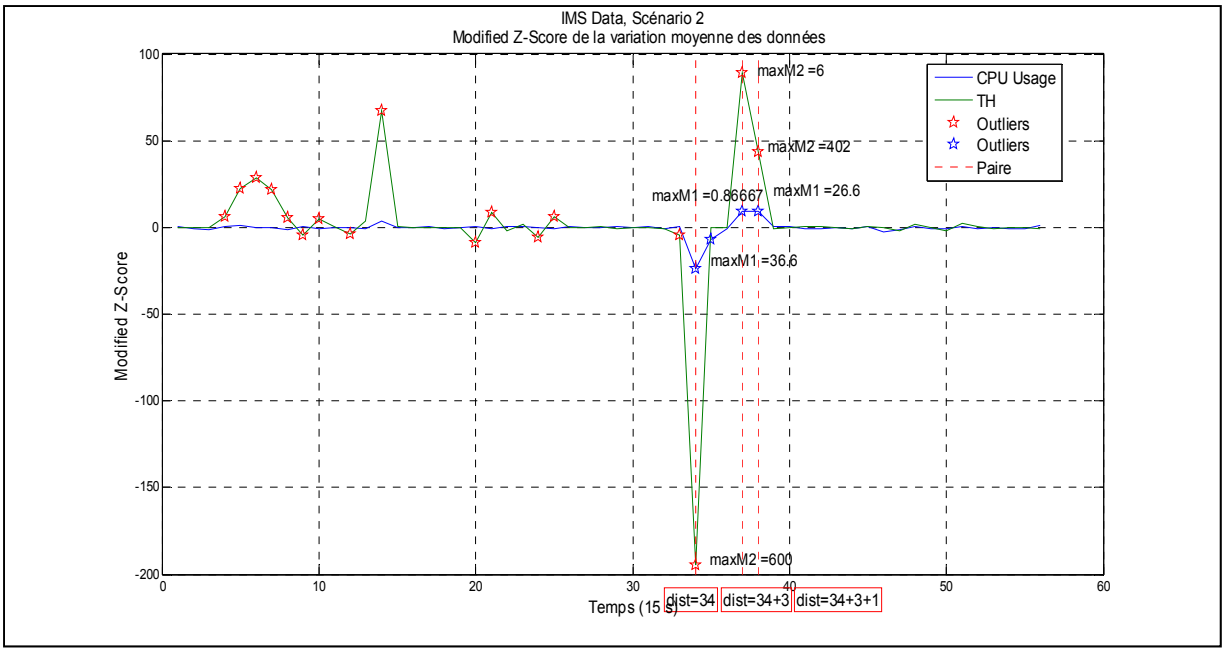

Figure 2.3 Scénario 2 - Valeur de Modified z-score CPU et TH

En faisant la correspondance entre les deux métriques étudiées CPU et TH, 3 paires d'outliers sont identifiées aux positions 34, 37 et 38, représentées dans la figure 2.3 par une ligne pointillée en rouge indiquant la position de chaque paire d'outliers. La variable « dist » indique la position des paires d'outliers dans le temps par rapport aux premières paires d'outliers détectées.

Nous avons respectivement 36.6, 0.8667 et 26.6 comme valeur de CPU. Nous remarquons sans la figure 2.4 que la valeur de CPU n'a pas dépassé le 50 %. Pour certains systèmes virtualisés, 50% de la consommation de CPU peut être une valeur critique pour les fournisseurs de services cloud. Dans notre cas, nous allons considérer toutes données qui s'approchent de la limite 50% comme une valeur critique. La méthode de Modified Z-score enregistre des outliers lorsqu'elle détecte des valeurs ayant une grande variation par rapport à l'ensemble de données. Une consommation de 37% pour le CPU n'est pas une valeur critique pour un système, mais peut nous indiquer un comportement anormal de la machine hébergeant ce système nécessitant un ajustement des ressources.

Nous pouvons constater que 2 outliers peuvent être identifiés comme de vrais outliers. Le premier à la position 34 et le deuxième à la position 38 (voir figure 2.3). Le premier outlier indique une diminution de la consommation de la ressource CPU et une diminution du throughput. En effet, le throughput passe de 600 à 6 m/s (message par seconde) et le CPU passe de 36.6% à 6 %. La deuxième grande variation des données a été détectée à la position 38 où le système reprend son fonctionnement normal. Les deux métriques CPU et Throughput enregistrent une baisse des valeurs de la position 34 jusqu'à la position 38. Ces variations ont été repérées par la méthode de détection d'outliers. La fiabilité de Modified Z-score pour ce scénario est F=2/3 =66%.

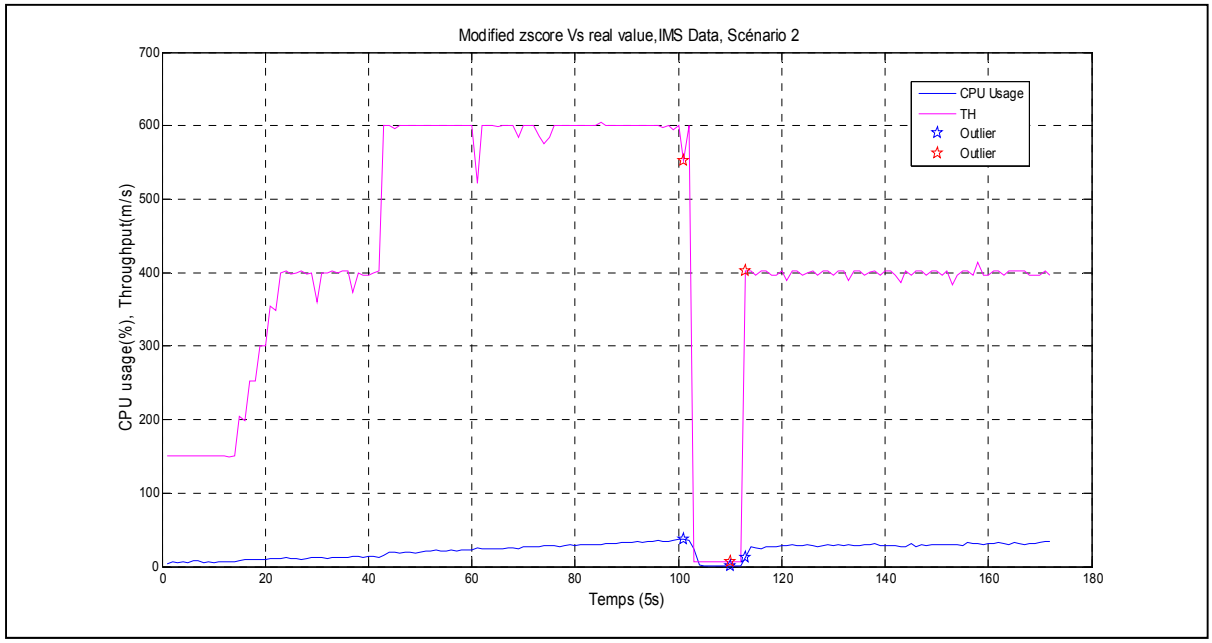

Figure 2.4 Scénario 2 - Variation des données réelles CPU et TH

2.4.1.3 Scénario 3

La valeur du Modified Z-score et la variation des données réelles pour le scénario 3 sont illustrées dans les figures 2.5 et 2.6.

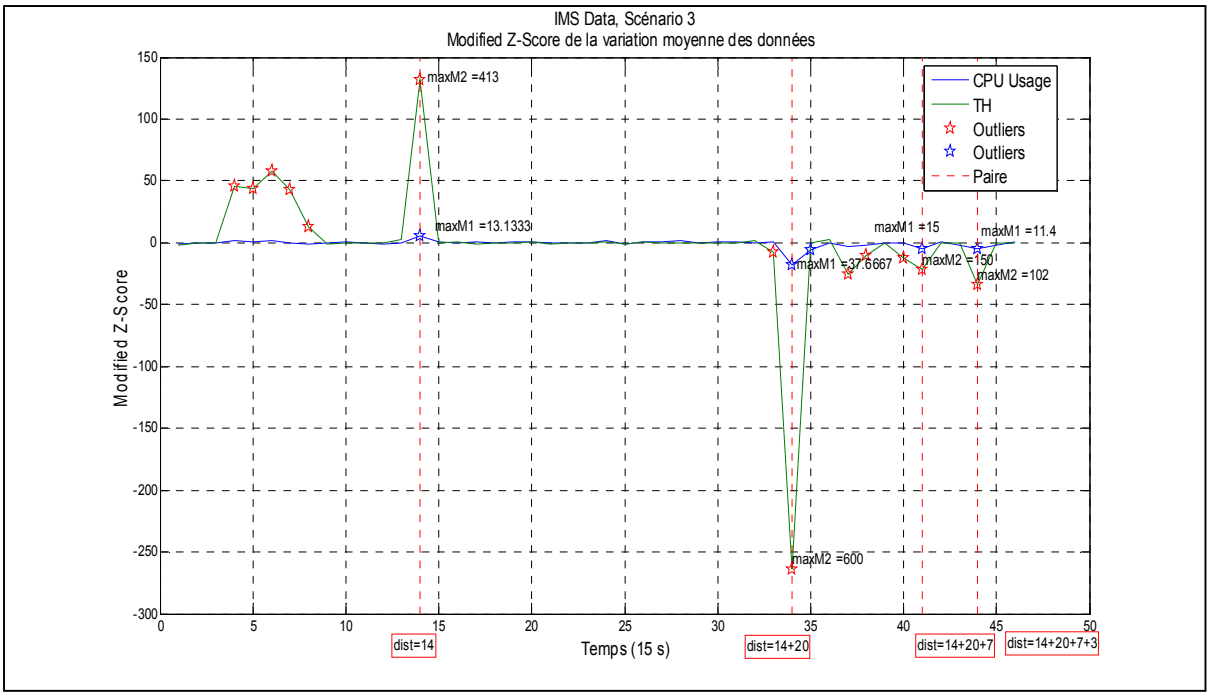

Figure 2.5 Scénario 3 - Valeur de Modified Z-score CPU et TH

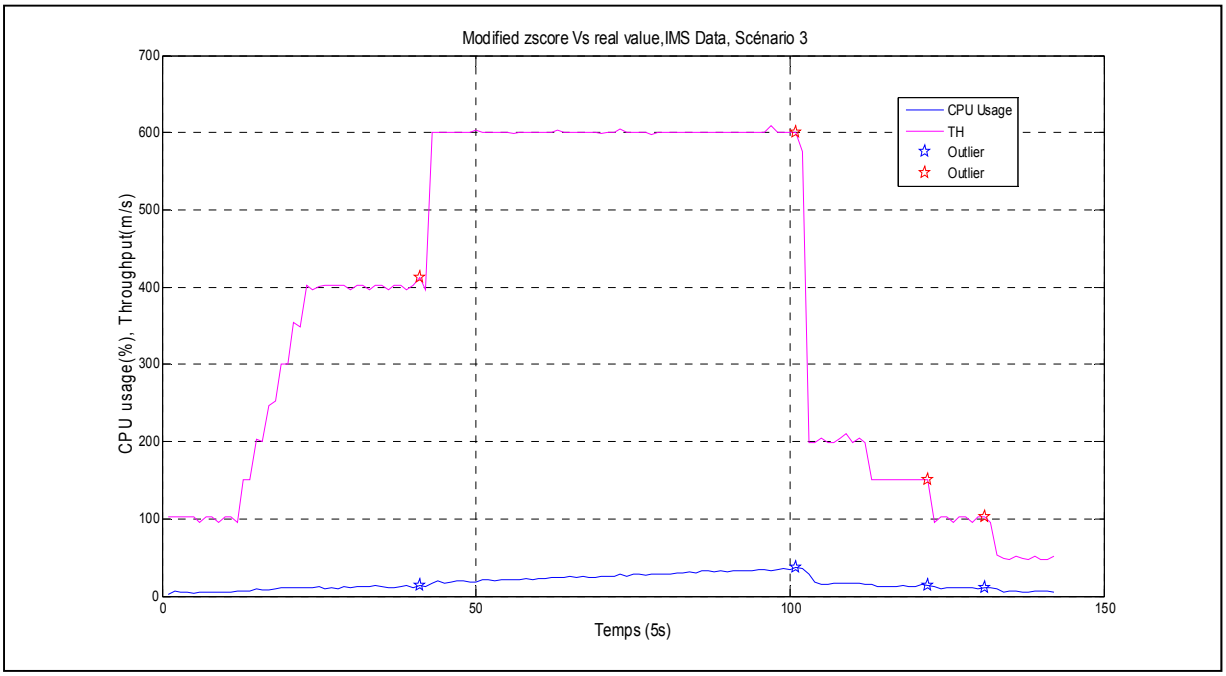

Figure 2.6 Scénario 3 - Variation des données réelles CPU et TH

Dans ce scénario, nous avons identifié 4 paires d'outliers représentés dans la figure 2.6 pour les deux métriques étudiées (CPU et TH). En regardant les deux courbes, on trouve une

correspondance entre les 2 métriques et surtout aux positions où les outliers sont observés. En effet, le CPU et le TH augmentent ou diminuent en même temps. De ce fait, on peut conclure qu'il y a un nombre important de messages échangés dans notre système virtualisé d'où le besoin d'ajuster les ressources en CPU.

La valeur maximale de CPU est égale à 37.66%. Cette valeur n'est pas critique et ne dépasse même pas la moitié des ressources disponibles. Cependant, les 4 paires de valeurs détectées ont été identifiées comme étant des outliers parce que leurs variations par rapport au reste des données sont importantes. La variation représentée dans la figure numéro 2.6 est le deuxième critère pour détecter et classifier les valeurs aberrantes. Donc pour voir si une valeur observée peut être considérée comme un outlier ou non, il faut tout d'abord voir sa valeur ensuite voir sa variation par rapport à l'ensemble des données. Cette variation nous indique les moments où l'approche Modified Z-score a enregistré de grands écarts dans la consommation de ressources et la throughput imposé au système/application virtualisé. Elle indique aussi le type d'adaptation de ressource que nous devons sélectionner à savoir : augmenter ou diminuer les ressources du système/application virtualisé.

Pour les 4 paires d'outliers détectés, seules les 2 premières paires peuvent être sélectionnées comme de vrais outliers. En effet, nous remarquons une grande variation par rapport au reste de données soit une augmentation brusque soit une diminution brusque du CPU et du TH. Cette variation est importante par rapport à la variation des 2 autres paires d'outliers.
Ainsi, la fiabilité pour ce scénario est F=2/4=50%.

2.4.1.4 Scénario 4

Dans le scénario 4, nous avons enregistré 4 paires d'outliers situées respectivement aux positions 14, 21 ,24 et 25 (voir figure 2.7). Les deux premiers outliers représentent une augmentation du CPU et du TH et les deux derniers une diminution graduelle. Nous pouvons nettement voir ces variations dans la figure 2.8 où nous avons superposé les outliers détectés aux données observées pour les deux métriques.

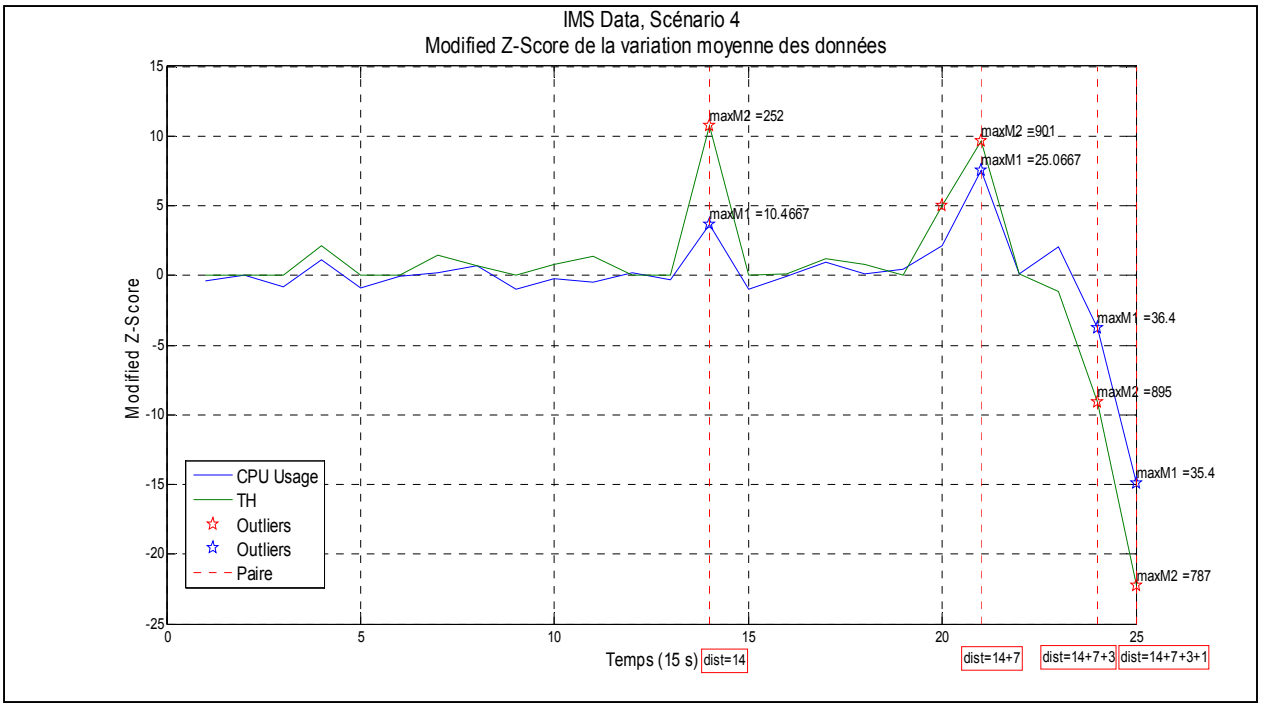

Figure 2.7 Scénario 4 - Valeur de Modified Z-score CPU et TH

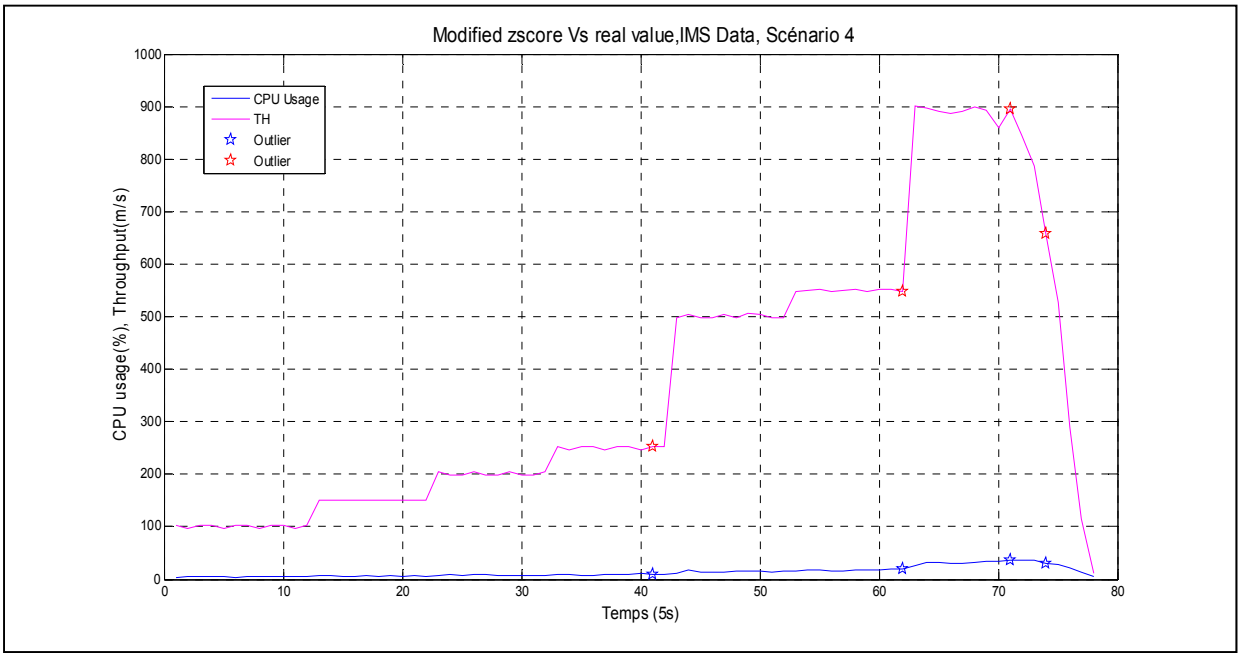

Figure 2.8 Scénario 4 - Variation des données réelles CPU et TH

La première paire d'outliers détectés nous indique une augmentation brusque du TH qui varie de 150 à 500 (m/s) avec une légère augmentation du CPU. Cette augmentation nous indique

que des messages sont en train d'être échangés dans notre système virtualisé. La deuxième paire d'outliers détectés présente aussi une augmentation des 2 métriques où le TH passe de 550 à 900 (m/s) ce qui indique que le nombre des messages a augmenté rapidement (dans un intervalle de temps de 5 secondes). La troisième paire d'outliers présente une diminution en termes de CPU et TH, cette diminution persiste jusqu'à la fin du test. Ces 3 paires d'outliers peuvent être considérées comme de vrais outliers puisque le flux des messages échangés est en train d'augmenter graduellement jusqu'à la détection du 3ème outlier où le flux de données circulant dans le réseau a diminué. La dernière paire d'outlier a été détecté à cause de la chute des valeurs de deux métriques. Cette baisse des valeurs a permis de détecter un outlier. Elle ne représente pas un comportement anormal du système, mais plutôt un besoin d'adapter les ressources pour éviter leur sous-utilisation. La fiabilité pour ce scénario est de ¾ =75%.

2.4.2 Analyse des données SMTP et Web

Dans cette partie, nous proposons d'analyser les données collectées de la plateforme cloud de l'ÉTS.

2.4.2.1 Scénario 5

Les données de l'ÉTS sont des données de type TI, le premier scénario de déploiement représente les ressources utilisées pour les échanges des courriels au sein de l'ÉTS. Dans ce scénario, nous avons détecté 5 paires d'outliers, représentés dans la figure 2.9 par les lignes pointillées en rouge.

Le premier outlier détecté pour le CPU dans la figure 2.9 ne peut pas être considéré comme un vrai outlier parce que sa valeur réelle est de 9% et sa variation est égale à -4,5275 ce qui indique une diminution dans l'utilisation de CPU et on remarque qu'elle augmente juste après. Les deux premiers outliers détectés pour la bande passante (BP), représentés aussi dans la figure 2.9, ne peuvent pas être considérés comme outliers non plus. Le seuil que nous avons fixé pour cette métrique est de 50% pour une capacité de 2 Mbps. La valeur de la métrique pour ces deux positions est égale à 624 KB/s et 974 KB/s et leurs variations sont de 4,2162 et -4,6493 respectivement. La valeur 974 KB/s est importante puisqu'elle est proche du seuil de 50% que

nous avons fixé pour la consommation de la bande passante. Ces deux outliers se suivent dans le temps et leurs sens de variation s'inverse, au début on remarque une augmentation suivit d'une diminution de l'utilisation de la bande passante, cependant on ne remarque pas des grandes variations en termes de CPU ce qui indique que la charge de trafic dans le système a augmenté, mais n'a pas affecté les ressources de CPU. C'est pour cette raison, on n'a pas besoin de traiter ces deux valeurs.

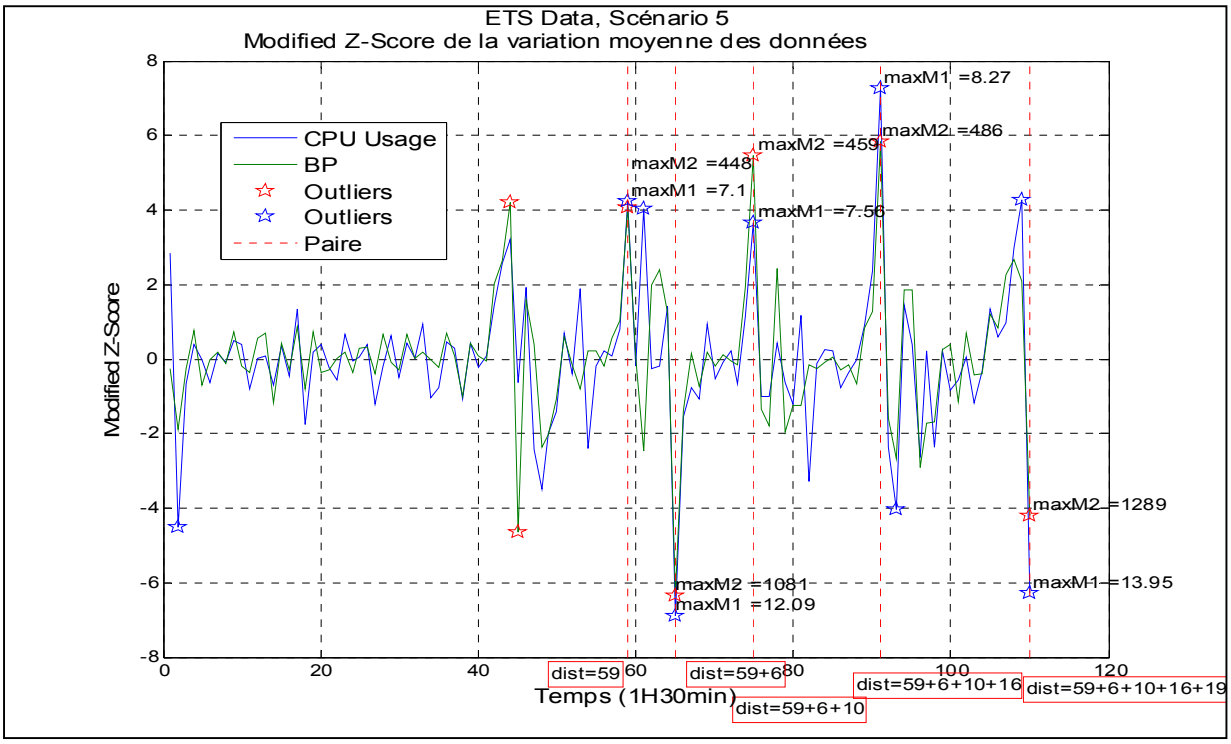

Figure 2.9 Scénario 5 - Valeur de Modified Z-score CPU et BP

Dans l'analyse qui suit nous intéressons à la correspondance entre les deux métriques étudiées, CPU et BP. On se retrouve avec 5 outliers pour le premier type d'applications « ETS_SMTP ». Le type d'application nous permet de savoir quelles sont les ressources les plus utilisées par cette application et nous permet de prioriser les métriques étudiées. Par exemple, pour le type d'application SMTP, la bande passante est la métrique la plus importante afin de garder une bonne qualité de service aux utilisateurs. Cependant, nous nous intéressons aussi au CPU pour maintenir une qualité de service pour les requêtes traitées.

Les autres outliers enregistrés pour le CPU dans ce scénario ne dépassent pas une valeur de 12% et leurs variations ne sont pas stables. Dans la figure 2.10, on remarque des variations assez fréquentes dans cet ensemble de données où elles augmentent puis elles diminuent puis elles augmentent de nouveau. En se basant seulement sur la valeur de CPU collecté on peut dire que le système n'aura pas besoin d'ajustement de ressources, cependant, si on prend en considération l'analyse des outliers collectés et la variation des valeurs collectées, on doit revoir et prévoir tous changement de comportement dans le système étudié.

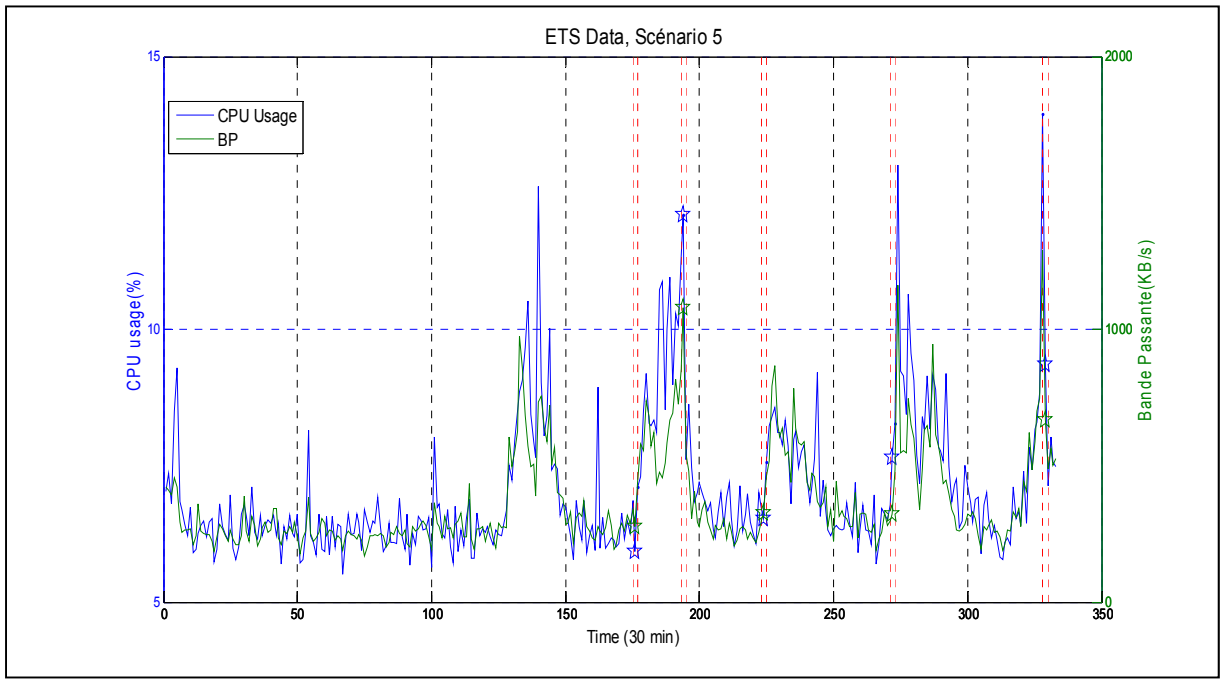

Figure 2.10 Scénario 5 - Variation des données réelles CPU et BP

Le tableau 2.4 montre les valeurs observées par rapport à la variation de ces 5 paires d'outliers pour les deux métriques en question. Dans ce tableau, on remarque que la valeur de la bande passante augmente alors que le sens de variation change, la bande passante passe de 448 KB/s à 1081 KB/s et la variation de 269.66 à -428.66, dans ce cas, le système n'a pas besoin de plus de ressource.

Tableau 2.4 Valeurs observées pour chaque outlier détecté versus leurs valeurs de variations

CPU		BP		Type d'outlier	
Valeur observée en %	**Valeur de la variation**	**Valeur observée en KB/s**	**Valeur de la variation**	**Vrai**	**Faux**
7.10	1.69	448	269.66		X
12.09	-2.70	1081	-428.66	X	
7.56	1.47	459	363.66		X
8.27	2.89	486	387	X	
13.95	1.71	1289	136.33	X	

Ce changement de signe nous indique qu'il y a eu une chute de la consommation de la métrique sélectionnée. Pour cette raison qu'on calcule la variation des trois valeurs observées à la fois afin de mieux voir les fluctuations de comportement du système. À la 3ème ligne du tableau 2.4 montre que la valeur observée de la métrique diminue alors que la valeur de la variation augmente et son sens change. Ceci peut être interprété par le comportement de système et la fluctuation des valeurs. La valeur de la variation nous donne une idée sur l'état des ressources, comme dans l'exemple précèdent, si on remarque une augmentation dans la valeur observée (ex. une augmentation de 448 KB/s à 1081 KB/s) tandis que dans la variation nous remarquons une diminution (ex. de 269.66 à -428.66), on peut conclure que le système à l'instant t+1 aura une diminution de la métrique concernée et n'a pas besoin de plus de ressource au contraire on peut en libérer. Cette décision peut être prise si on remarque que la diminution de la valeur de variation est enregistrée pour une certaine période de temps et que le système n'a pas repris son comportement habituel. La fiabilité de Modified Z-score pour ce scénario est de 60% (F= 3/5).

2.4.2.2 Scénario 6

Dans ce qui suit, nous analysons les résultats de Modified Z-score appliqué sur les données web de l'infrastructure cloud de l'ÉTS.

La figure 2.11 représente la valeur de Modified Z-score et les outliers détectées pour ce scénario. La figure 2.12 illustre les outliers détectés et leurs positions dans les données observées.

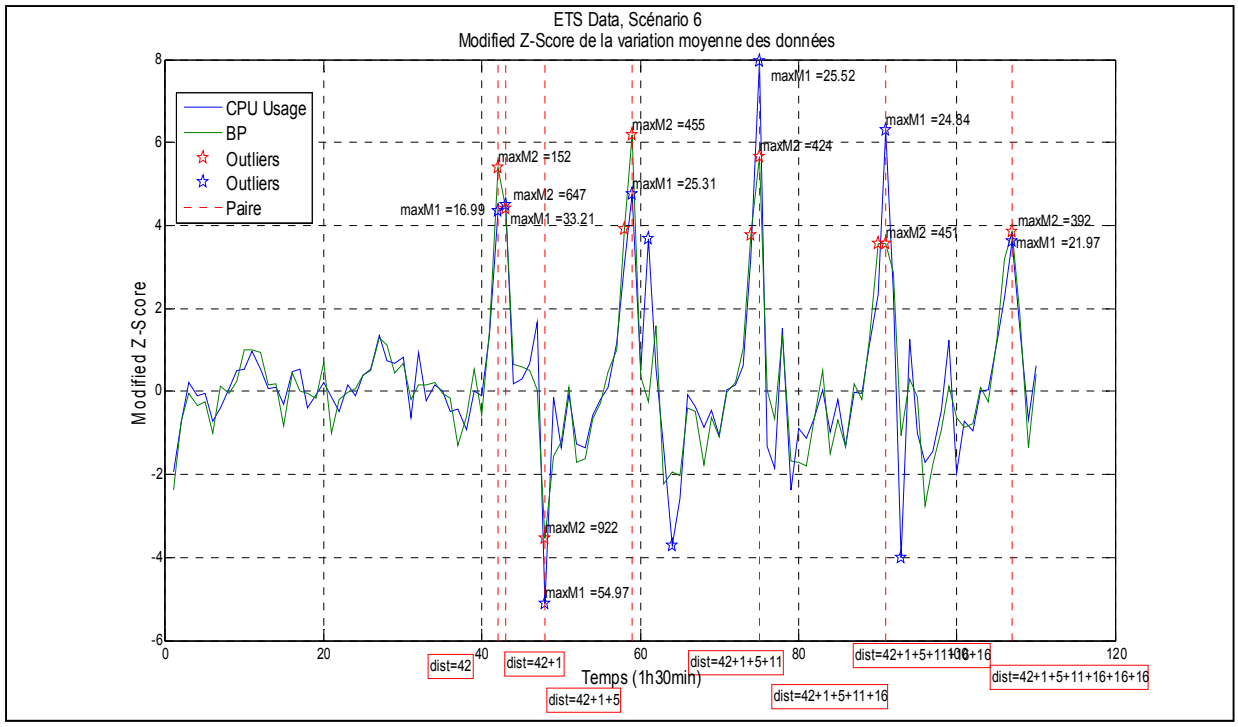

Figure 2.11 Scénario 6 - Valeur de Modified Z-score CPU et BP

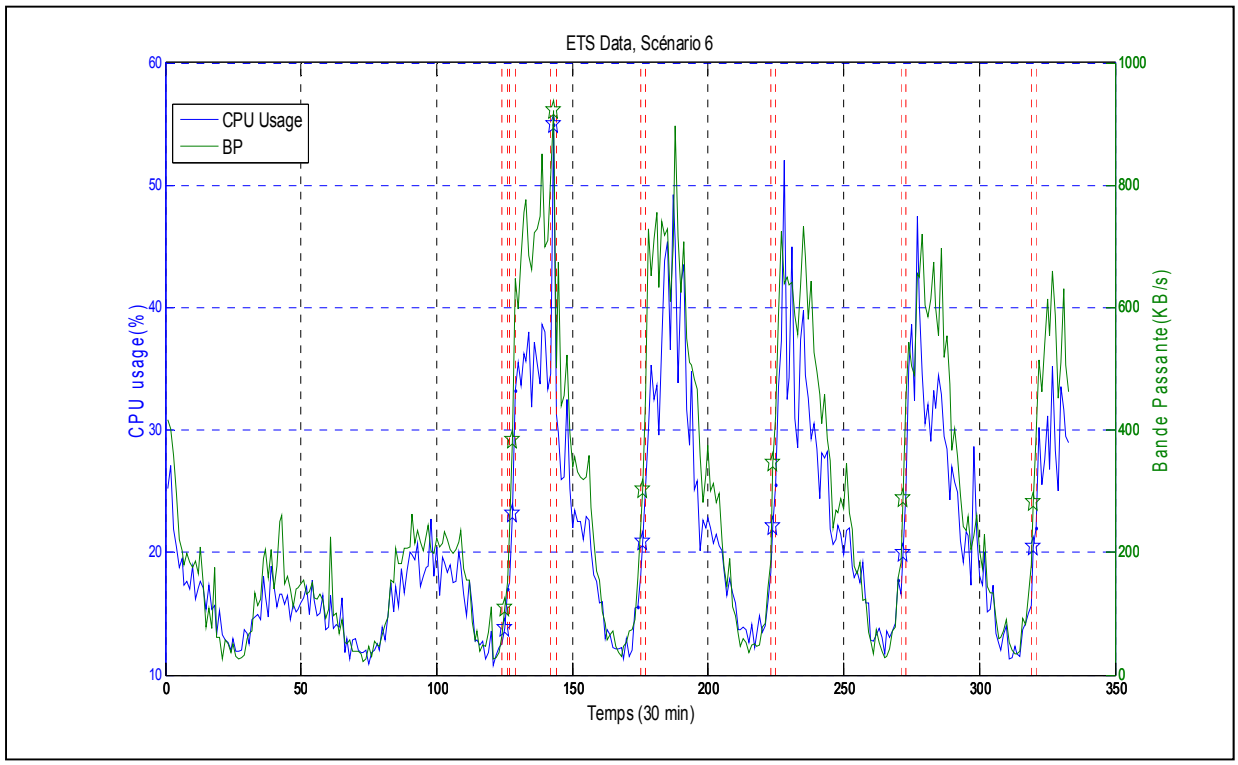

Figure 2.12 Scénario 6 - Variation des données réelles CPU et BP

Avec la méthode de Modified Z-score, on peut extraire 7 paires d'outliers pour les deux métriques CPU et bande passante pour le web.

Le tableau 2.5 montre les valeurs observées par rapport à la variation de ces outliers. Dans ce tableau, on remarque une augmentation progressive de la valeur de CPU, la valeur de variation correspondante augmente au début puis elle diminue. Le premier outlier est un vrai outlier puisqu'il nous indique une augmentation en termes de ressources, cependant le deuxième outlier est un faux outlier puisqu'il ne reflète pas une grande variation même si on remarque une augmentation dans les valeurs de deux métriques. Le troisième outlier enregistre une augmentation de 20% pour le CPU et de 300 KB/s pour la bande passante. Cette augmentation est accompagnée d'une variation négative ce qui nous indique qu'on peut réduire la quantité de ressources déployées pour éviter la sous-utilisation de celles-ci. Finalement, les 4 derniers outliers sont des vrais outliers puisqu'ils enregistrent des augmentations dans différents moments de l'exécution des services. Ces augmentations et ces diminutions dans les 2 métriques nous indiquent des changements dans l'utilisation des applications web du système virtualisé de l'ÉTS, soient le nombre des requêtes envoyées et reçues dans le cloud. La fiabilité de ce scénario est de 86% où F=6/7.

Tableau 2.5 Valeurs observées pour chaque outlier détecté versus leurs valeurs de variations

CPU		**BP**		**Type d'outlier**	
Valeur réelle en %	**Valeur de la variation**	**Valeur réelle en KBps**	**Valeur de la variation**	**Vrai**	**Faux**
16,99	10,17	152	315	X	
33,21	10,513	647	255,33		X
54,97	-12,96	922	-222,66	X	
15,58	7,063	159	226	X	
25,52	18,97	424	331,66	X	
17,73	5,28	159	204,66	X	
21,97	8,43	392	222	X	

2.4.2.3 Scénario 7

Dans cette section, nous analysons les résultats du scénario 7 pour le type d'application « ETS_SQL ».

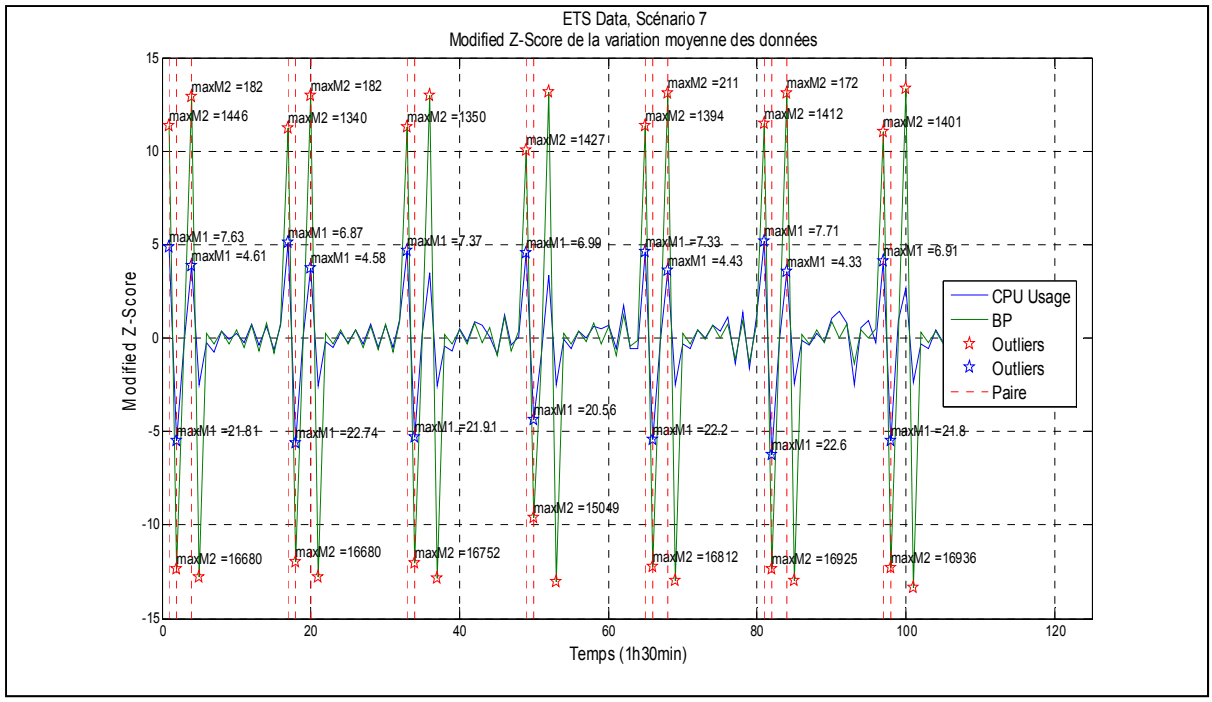

Figure 2.13 Scénario 7 - Valeur de Modified Z-score CPU et BP

Dans la figure numéro 2.13, on remarque que les données sont cycliques et les outliers détectés se répètent dans chaque cycle. Les cycles se répètent après 13 ou 15 unités de temps. Il y a trois cycles où on trouve moins d'outliers que les autres spécifiquement pour la métrique CPU, ce sont le troisième, le quatrième et le septième cycle. Dans ces trois cycles les variations de CPU font que l'outlier n'est plus affiché. La valeur du CPU et sa variation ne sont pas importantes. C'est pour cette raison qu'aucun outlier dans ces cycles n'a été identifié.

Les outliers détectés dans chaque cycle se répètent. On remarque une correspondance entre les 2 premiers outliers des 2 métriques dans chaque cycle. Cependant, le troisième outlier détecté reflète une augmentation en bande passante et une diminution de CPU, donc le système n'a pas besoin de ressource et on peut le considérer comme faux outlier. Ces variations sont représentées dans la figure 2.14 où on peut conclure qu'il y a une correspondance dans l'allure de données de deux métriques. Le premier outlier de chaque cycle reflète une augmentation dans les 2 métriques étudiées et le deuxième outlier montre une diminution dans les deux métriques c'est pourquoi on les considère comme vrais outliers. Cette augmentation est nettement plus visible dans les données observées représentées dans la figure 2.14.

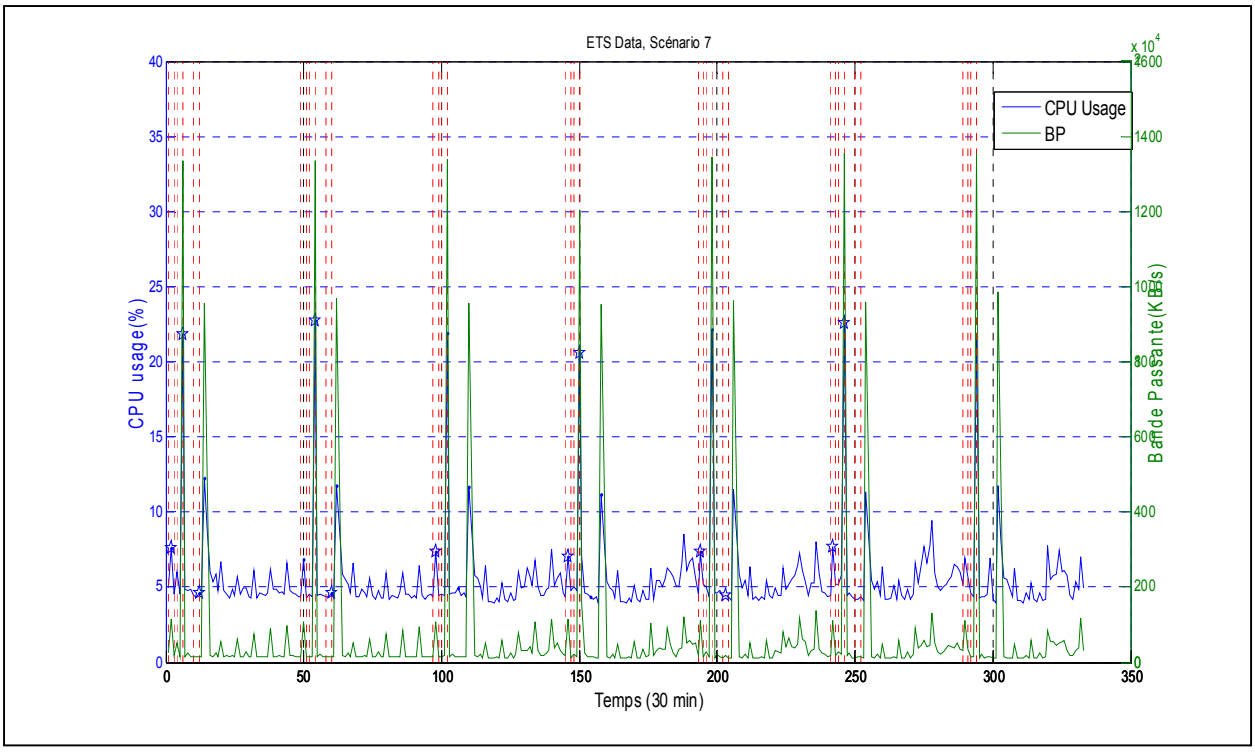

Figure 2.14 Scénario 7 - Variation des données réelles CPU et BP

Le tableau 2.6 représente les vrais et les faux outliers détectés pour les deux métriques CPU et BP. La fiabilité de Modified Z-score pour ce scénario est de 78% où F=14/18

Tableau 2.6 Valeurs observées pour chaque outlier détecté versus leurs valeurs de variations

CPU		BP		Type d'outlier	
Valeur réelle en %	**Valeur de la variation**	**Valeur réelle en KB/s**	**Valeur de la variation**	**Vrai**	**Faux**
7,63	5,083	1446	5173	X	
21,81	-5,92	16680	-5617,33	X	
4,61	4,023	182	5877,33		X
12,24	-2,75	11949	-5826,66	X	
6,87	5,36	1340	5113,33	X	
22,74	-6,04	16680	-5462,33		X
4,58	3,88	182	5891,33	X	
11,77	-2,81	12131	-5832	X	
7,37	4,9	16812	5132	X	
21,91	-5,72	1350	-5493,33	X	
4,96	3,6	16752	5901	X	
11,71	-2,82	196	-5867,67	X	

Tableau 2.6 Valeurs observées pour chaque outlier détecté versus leurs valeurs de variations (Suite)

CPU		BP		Type d'outlier	
Valeur réelle en %	**Valeur de la variation**	**Valeur réelle en KB/s**	**Valeur de la variation**	**Vrai**	**Faux**
6,99	4,76	11961	4582		X
20,56	-4,75	1427	-4365,67	X	
4,39	3,46	15049	5974,33	X	
11,19	-2,78	187	-5952,67		X
7,33	4,84	11929	5167,33	X	
22,20	-5,86	1394	-5575	X	

2.4.3 Analyse des données de Google

Dans cette sous-section, nous allons analyser la méthode de Modified z-score avec les données de Google.

2.4.3.1 Scénario 8

Le scénario 8 représente la tâche numéro 8 extraite des données du cluster de Google (Wilkes, 2011) qui est considéré comme un service cloud. On prend les 2 métriques CPU et Temps d'exécution (TE) comme variables à étudier. Nous remarquons dans la figure 2.15 que la courbe de temps d'exécution n'est pas présente. Étant donné que les variables à étudier doivent suivre une distribution normale, le temps d'exécution ne peut pas être calculé ni affiché dans le graphique avec le Modified Z-score si la valeur collectée est constante. En fait, Google collecte les métriques chaque 5 min, c'est-à-dire pour une tâche qui s'exécute pendant 20 minutes, il collecte les résultats de son utilisation de CPU ou de mémoire chaque 5 min (300 secondes).

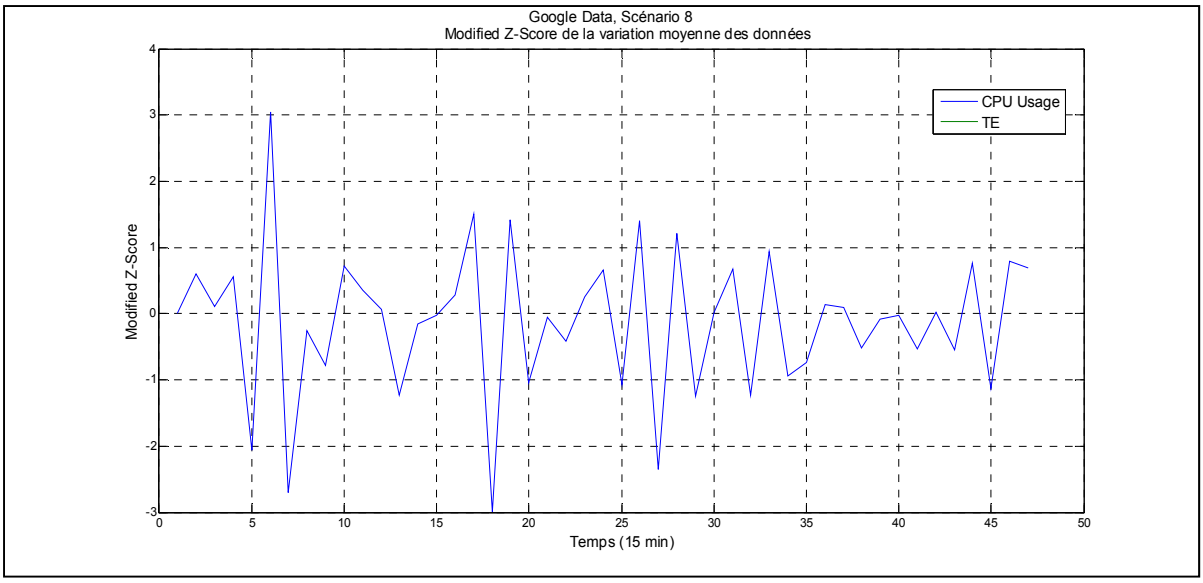

Figure 2.15 Scénario 8 - Valeur de Modified Z-score CPU et TE

Nous remarquons qu'avec le Modified Z-score nous n'avons pas enregistré des outliers pour les données de Google pour la tâche numéro 8. En représentant le temps d'exécution dans la figure 2.16 nous remarquons que cette valeur n'est pas constante tout le temps parce que la tâche en question a été exécutée en moins de temps (moins de 5 min) pendant certaines périodes de ce test.

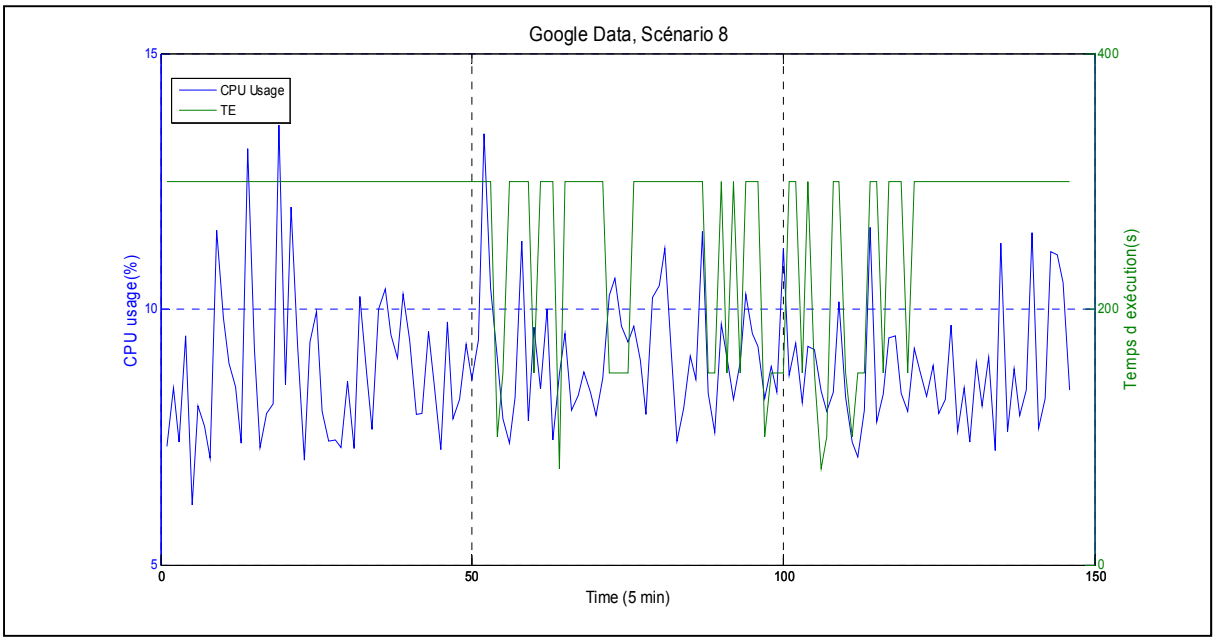

Figure 2.16 Scénario 8 - Variation des données réelles CPU et TE

2.4.3.2 Scénario 9

Pour la tâche Google numéro 26, nous remarquons dans la figure 2.17 l'existence des outliers pour les 2 métriques, mais pas de correspondance entre ces outliers détectés. Dans ce cas, il peut y avoir des mises à jour qui sont en train d'être exécuté où ils nécessitent une consommation importante de CPU. Ainsi, ce phénomène peut être expliqué par la défaillance du hardware ou un problème dans l'infrastructure cloud.

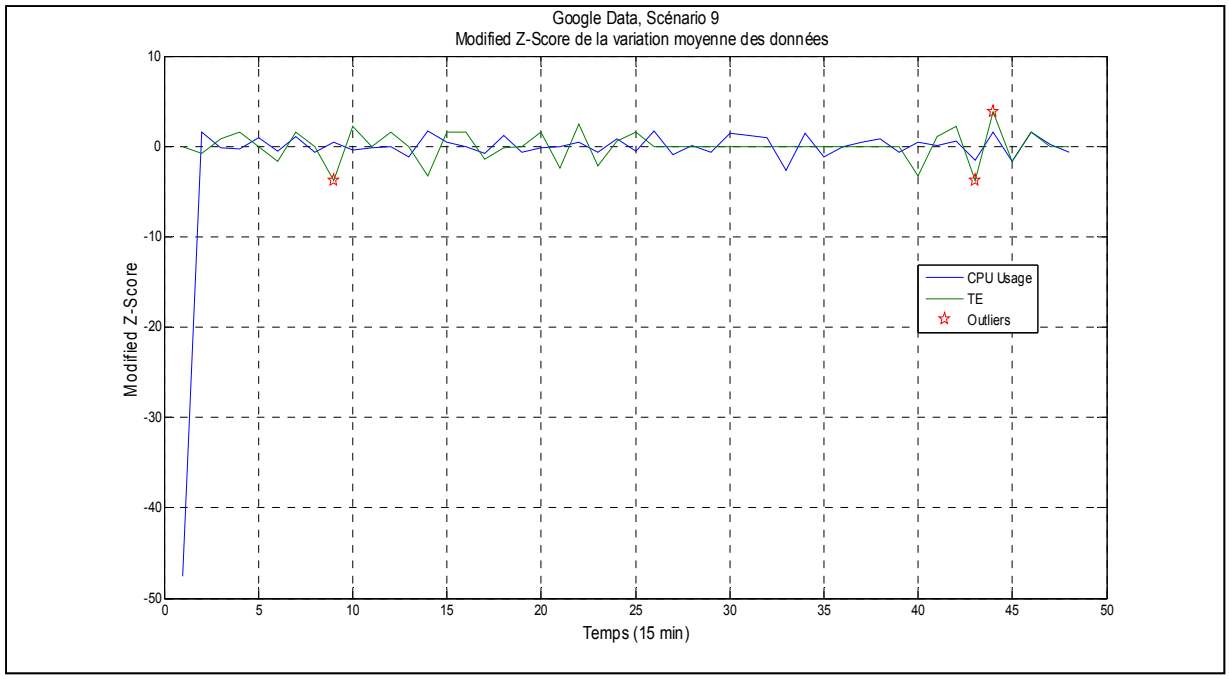

Figure 2.17 Scénario 9 - Valeur de Modified Z-score CPU et TE

Nous remarquons d'après la figure 2.18 que la valeur du temps d'exécution est variable c'est pourquoi son score a été représenté dans la figure 2.17.

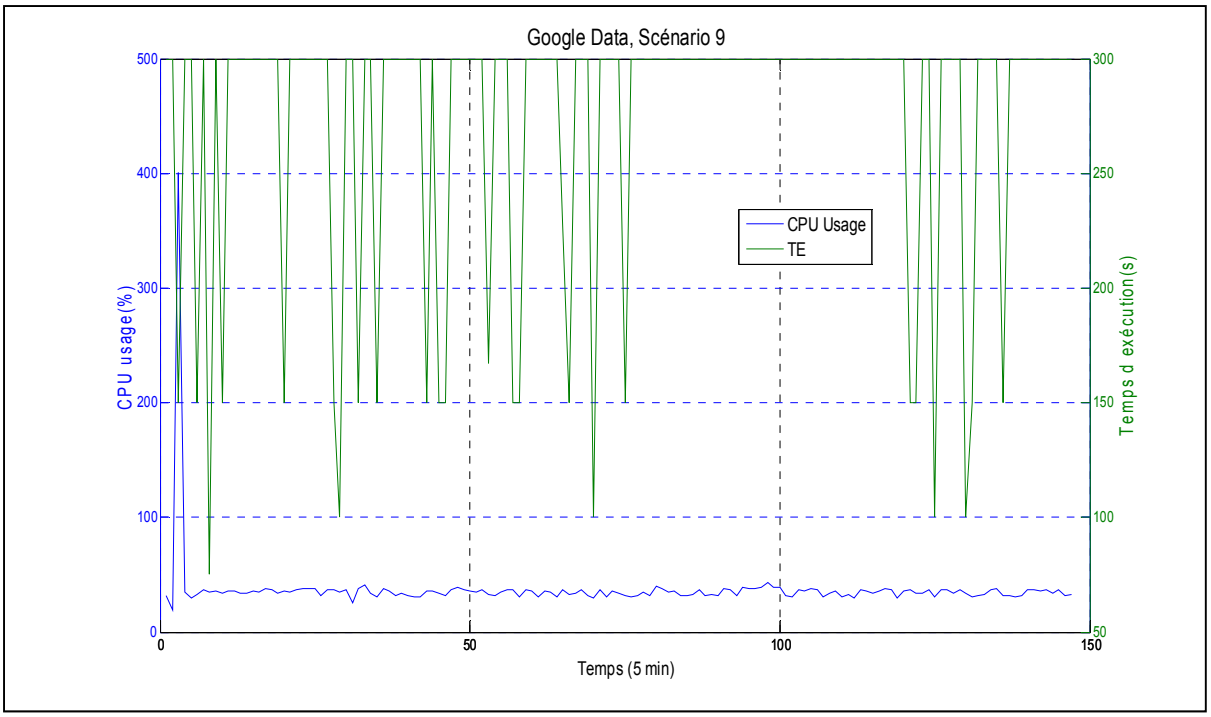

Figure 2.18 Scénario 9 - Variation des données réelles CPU et TE

2.4.3.3 Scénario 10

Dans ce scénario nous représentons la tâche numéro 42 de Google. Nous remarquons dans la figure 2.19 l'apparition de deux outliers pour la métrique CPU. Cependant, il n'y a pas de correspondance entre les deux métriques étudiées.

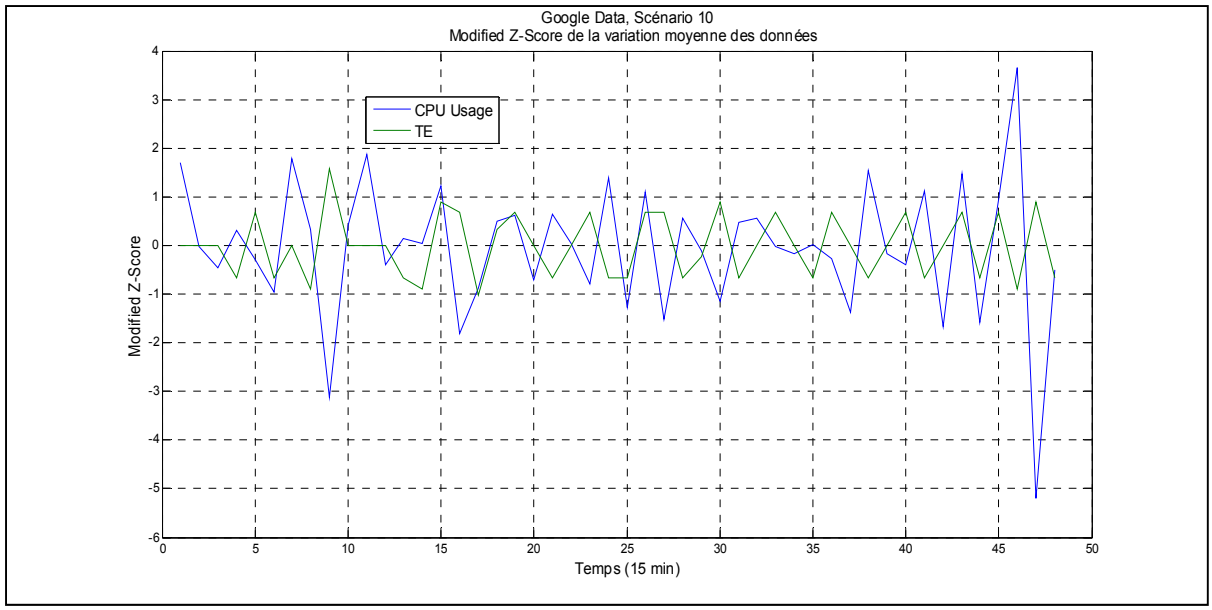

Figure 2.19 Scénario 10 - Valeur de Modified Z-score CPU et TE

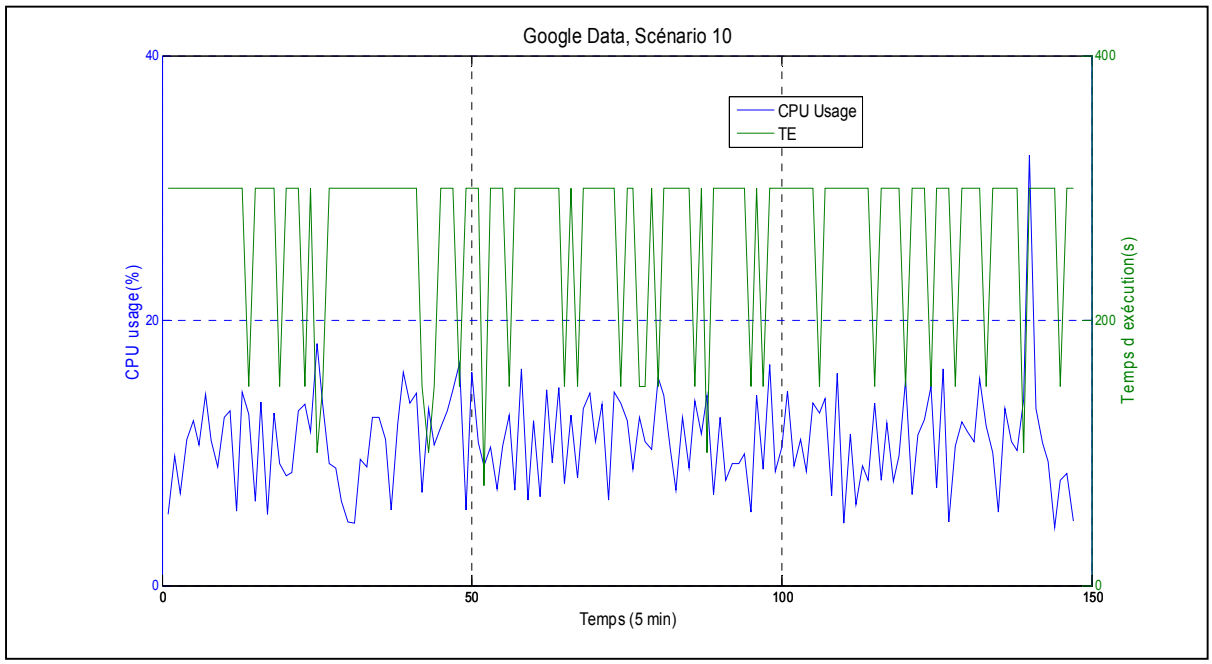

Figure 2.20 Scénario 10 - Variation des données réelles CPU et TE

Pour le premier outlier détecté, nous remarquons que le CPU augmente tandis que le temps d'exécution diminue ce qui peut être expliqué par le fait que la tâche a eu besoin de plus de CPU pour terminer son exécution. Pour le deuxième outlier, le CPU diminue et le TE augmente. On peut conclure que la tâche a mis plus de temps pour s'exécuter, mais ne nécessite

pas forcément beaucoup de ressources. L'allure de deux métriques dans le temps est représentée dans la figure 2.20.

2.4.3.4 Scénario 11

Dans ce scénario de déploiement, nous présentons la tâche numéro 57 de Google. La figure 2.21 montre le résultat des outliers détectés pour la métrique CPU. La deuxième métrique ne figure pas dans le résultat de Modified Z-score parce qu'elle ne suit pas une distribution normale. L'allure de de la courbe de temps d'exécution est présentée dans la figure 2.22. En comparant la valeur des outliers du CPU avec les valeurs relevées du temps d'exécution, nous pouvons identifier 11 outliers. Les outliers détectés pour la métrique CPU est 20, cependant, en superposant ces outliers avec leurs correspondants dans la métrique TE, nous pouvons détecter que 11 outliers

La plupart des outliers identifiés n'ont pas une valeur importante en CPU. La valeur maximale de CPU enregistré est de 50%. Les observations détectées ne sont pas seulement celles qui enregistrent les plus grandes valeurs par rapport à l'ensemble de données, mais aussi ceux ayant de grandes variations. Ainsi, pour identifier une donnée comme un outlier, nous lui associons d'abord la valeur de temps d'exécution et ensuite nous analysons le comportement de la tâche étudiée.

Seuls le cinquième outlier et le dernier ne peuvent pas être considéré comme des vrais outliers parce qu'en regardant les valeurs observées ces outliers n'indiquent pas un besoin d'adaptation de ressources pour le système. On remarque que même après une augmentation de la variation du CPU, elle tend à diminuer après.Ainsi, il faut toujours analyser les valeurs observées (voir figure 2.22) avant de prendre une décision d'adaptation des ressources. La fiabilité de Modified Z-score est de 82% où F=9/11.

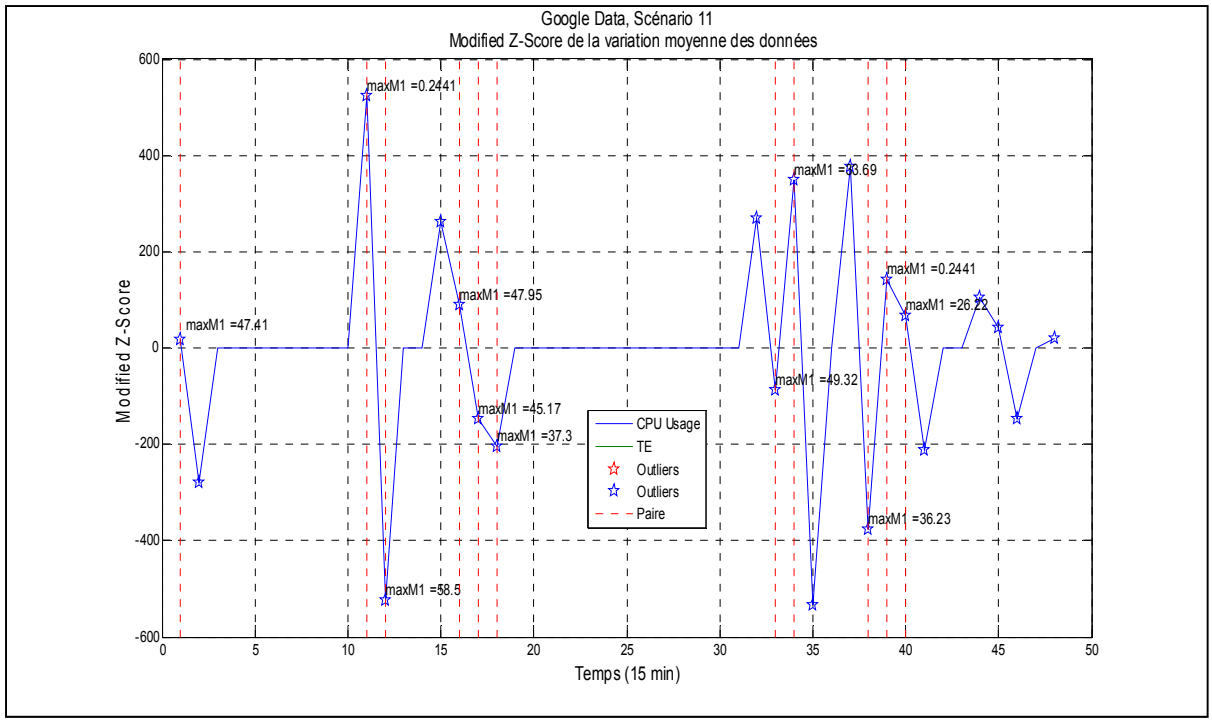

Figure 2.21 Scénario 11 - Valeur de Modified Z-score CPU et TE

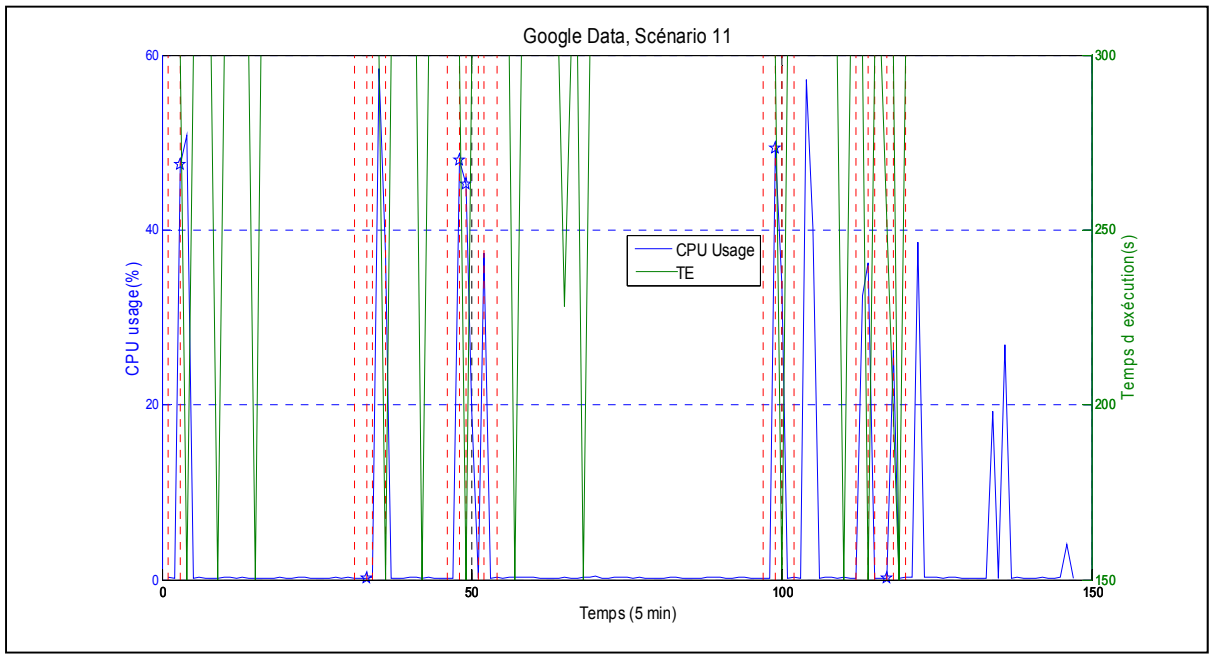

Figure 2.22 Scénario 11 - Variation des données réelles CPU et TE

2.5 Observations

Dans cette section, nous présentons un résumé des différentes observations tirées à partir de l'analyse précédente pour la méthode Modified Z-score. Le tableau 2.7 résume la fiabilité de la méthode pour chaque scénario mentionné dans la section 2.3. En moyenne, la fiabilité globale du Modified Z-score est de 69 % pour l'ensemble des données étudiés.

Tableau 2.7 Performance de l'approche Modified Z-score

Modified Z-score			**Remarques**
IMS	Scénario 1	-Pas d'outliers détectés	Pour les données d'IMS, les outliers détectés sont des vrais outliers lorsque le nombre des messages échangés dans le réseau est important et la consommation de CPU augmente ou diminue au même temps.
	Scénario 2	Fiabilité de 67%	
	Scénario 3	Fiabilité de 50%	
	Scénario 4	Fiabilité de 75%	
ETS	Scénario 5	Fiabilité de 60%	Les données de l'ÉTS varient beaucoup contrairement à celle d'IMS, c'est pour cette raison que nous enregistrons plus d'outliers dans ces scénarios.
	Scénario 6	Fiabilité de 86%	
	Scénario 7	Fiabilité de 78%	
Google	Scénario 8	Pas d'outliers détectés	La méthode Modified Z-score n'est pas adapté aux données comme celle de Google. Nous n'avons pas pu établir une correspondance entre les deux métriques pour les 3 scénarios 8, 9 et 10.
	Scénario 9	Pas d'outliers détectés	
	Scénario 10	Pas d'outliers détectés	
	Scénario 11	Fiabilité de 82%	

Conclusion

Dans ce chapitre, nous avons analysé la méthode de Modified z-score. Nous avons détecté plusieurs valeurs aberrantes dans les différents systèmes étudiés et nous avons pu déduire les vrais outliers. Nous avons obtenu une fiabilité de 69 % pour l'ensemble des données étudiées.

Dans le prochain chapitre, nous allons analyser une autre approche de détection des outliers appelée Mahalanobis. Nous allons la comparer avec l'approche Modified Z-score. Nous allons confronter les fiabilités des deux approches pour ne retenir qu'une seule qui sera utilisée dans l'algorithme d'adaptation de ressources.

CHAPITRE 3

DÉTECTION DES OUTLIERS BASÉE SUR LA DISTANCE MAHALANOBIS

Introduction

Plusieurs méthodes sont utilisées pour analyser les comportements des systèmes virtualisés. Cependant, avant que les données, extraites de ces systèmes, ne puissent être utilisées, il est important de détecter les valeurs aberrantes sur l'ensemble de ces données. Les méthodes classiques de détection des valeurs aberrantes ne sont pas toujours efficaces, simplement parce qu'elles sont affectées par les données elles-mêmes.

Par exemple, l'approche Modified Z-score perd de son efficacité à cause des problèmes de masquage et de submersion causée par l'impact des valeurs aberrantes sur les données. Dans ce chapitre, nous proposons d'analyser la distance Mahalanobis qui empêche l'influence de certains outliers sur l'ensemble des données. Cette dernière nous permet de mettre le point sur ces phénomènes tout en nous basant sur les données et sur la corrélation entre ces différentes valeurs étudiées.

3.1 Description de l'approche

3.1.1 Définitions

Avant de décrire l'approche proposée, nous allons définir les termes et les symboles qui seront utilisés. Les symboles sont résumés dans le tableau 3.1 alors que les définitions sont listées dans ce qui suit.

Tableau 3.1 Tableau des symboles utilisés

Symbole	Description
D_M	Résultat de la distance Mahalanobis (La valeur de la MD)
$x = (x_1, x_2, x_3, \ldots, x_p)^T$	Vecteur à plusieurs variables
$\mu = (\mu_1, \mu_2, \mu_3, \ldots, \mu_p)^T$	Ensemble de vecteurs de valeurs moyennes
$\sum$	Matrice de covariance
$\sum^{-1}$	Inverse de la matrice de covariance

Tableau 3.1 Tableau des symboles utilisés (Suite)

Symbole	Description
$\sigma(y,x) = \sigma(x,y)$	Covariance de x et y
$\sigma(x,x)$	Variance de x
$\sigma(y,y)$	Variance de y
λ	Valeur scalaire appelée 'valeur propre'.
I	Matrice identité de la matrice $\sum$.

- **La distance Mahalanobis (*Mahalanobis Distance* - MD) :** La distance Mahalanobis a été introduite en 1936 par Prasanta Chandra Mahalanobis. Cette distance est basée sur la corrélation entre les variables étudiées. Dans ce travail, ces variables représentent les métriques collectées dans les différents environnements de test. Elle permet de déterminer la similarité entre un ensemble de données connues et inconnues. À la différence de la distance euclidienne, cette méthode prend en compte la variance et la corrélation de l'ensemble de données. La distance de Mahalanobis accorde un poids moins important aux composantes les plus dispersées tandis que la distance euclidienne traite indépendamment et de la même façon toutes les composantes des vecteurs de données. (Mahalanobis, 1936)

En considérant la covariance des données et la variation des différentes variables, la distance de Mahalanobis, est utile pour détecter les valeurs aberrantes pour différentes données collectées de différents systèmes. Pour calculer la MD il faut utiliser l'équation suivante : (Mahalanobis, 1936)

$$D_M(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)} \qquad (3.1)$$

Où :

$D_M(x)$= la valeur de la distance Mahalanobis de x.

$x = (x_1, x_2, x_3, \ldots, x_p)^T$: un vecteur à plusieurs variables, dans notre cas, différentes métriques (ex., CPU, Bande passante, etc.).

$\mu = (\mu_1, \mu_2, \mu_3, \ldots, \mu_p)^T$: un ensemble de vecteurs de valeurs moyennes de x.

$\sum$ = Matrice de covariance

$\sum^{-1}$ = Inverse de la matrice de covariance

La distance de Mahalanobis est basée à la fois sur la moyenne et la variance des variables et aussi sur la covariance entre ces variables. La région qui représente la distance constante de Mahalanobis autour de la moyenne forme un ellipsoïde lorsque 2 variables sont mesurées, ou un hyper-ellipsoïde lorsque l'on utilise davantage de variables. La probabilité associée à chaque ellipse suit une distribution du khi-deux avec p degrés de liberté et une probabilité de 1-α.

$$(x-\mu)^T \Sigma^{-1} (x-\mu) \leq \chi_p^2(\alpha) \qquad (3.2)$$

La distribution du khi-deux permet de vérifier l'adéquation d'une distribution empirique à une loi de probabilité donnée. Plus généralement, elle s'applique dans le test d'hypothèses avec certains seuils. Elle est également utilisée pour établir des intervalles de confiance concernant la variance ou l'écart-type de variables aléatoires.

La figure 3.1 illustre l'ellipse de Mahalanobis, la distance entre le centroïde et les différentes variables. Les variables qui se trouvent sur la même ellipse, sont équidistantes. On peut définir plusieurs ellipses pour représenter la distance Mahalanobis, et ce en changeant l'intervalle de confiance (probabilité). Cet intervalle est défini à la section 3.2.1.

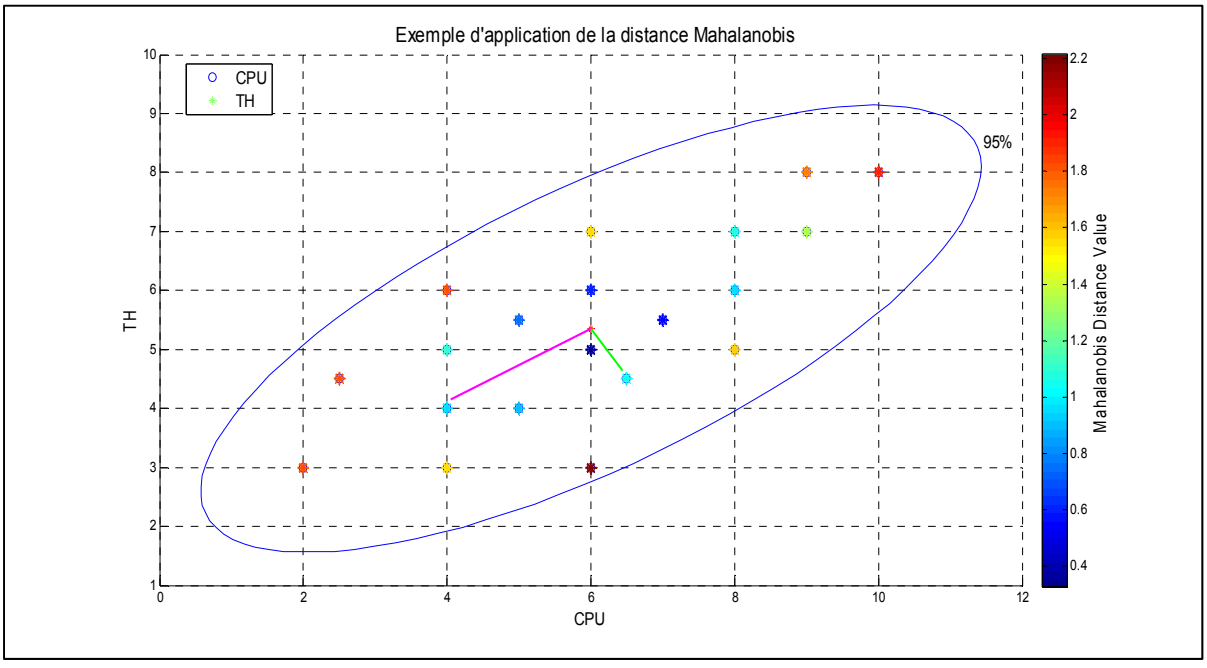

Figure 3.1 Ellipsoïde de la distance de Mahalanobis pour les deux métriques CPU et TH

- **La valeur critique**

La valeur critique est la valeur permettant d'identifier un outlier dans l'ensemble des données. Toute donnée supérieure à cette valeur est considerée comme une valeur aberrante. Elle est déterminée en prenant en compte la valeur de probabilité définie et les degrés de liberté. Ainsi, si 2 variables sont analysées (ex., CPU et TH), alors le degré de liberté = 2. Par exemple, si p=2 et α=0,5 nous avons $\chi^2 = 5.99$ avec une probabilité de 0,95. L'équation 3.2 montre que 5% du carré de la distance de Mahalanobis (DM^2) doit être supérieur ou égal à 5.99 . La racine carrée de la valeur critique χ^2 est utilisée comme un seuil afin de déterminer les outliers qui représentent des valeurs aberrantes pour les variables analysées. Cette valeur permet de calculer les axes de l'ellipse de confiance (voir la section 3.2.2). Pour illustrer une ellipse avec cette valeur critique, nous devons calculé le. $\sqrt{\chi^2}$.

$$\sqrt{\chi^2} = \sqrt{5.99}=2.45 \tag{3.3}$$

Si l'analyse se porte sur 3 variables, donc le degré de liberté = 3, et ainsi de suite. Ainsi, les degrés de liberté pour notre cas sont égaux au nombre de variables étudiées. Tout score de distance Mahalanobis au-dessus de cette valeur critique est considéré comme outlier. (Johnson et Wichern, 2007)

- **La matrice de covariance**

La matrice de covariance connue aussi sous le nom de matrice de dispersion ou la matrice de variance-covariance notée $\sum$, généralise la notion de variance en plusieurs dimensions. Elle permet de calculer et d'évaluer la variation de chaque variable par rapport aux autres variables.

La matrice de covariance a 2 variables x et y. Dans ce projet, ces deux variables représentent les deux métriques étudiées (ex., CPU et TH) . Elle est définie comme suit : (Mahalanobis, 1936)

$$\Sigma = \begin{bmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{bmatrix} \tag{3.4}$$

Pour un ensemble de p variables aléatoires réels x_1, ….., x_p , cette matrice est une matrice carrée dont la valeur de la ligne *i* et de la colonne *j* est la covariance des deux variables x_i et x_j. Parce que la covariance de la ième variable aléatoire avec le jème variable est la même que la covariance du jème variable aléatoire avec la ième variable, chaque matrice de covariance est symétrique et positif semi-définie.

La covariance de la ième variable aléatoire avec elle-même est simplement la variance de cette variable, chaque élément de la diagonale principale de la matrice de covariance est la variance d'une des variables. Par conséquent, la matrice de covariance est toujours une matrice symétrique avec les variances sur sa diagonale et les covariances hors de sa diagonale. Cette matrice nous indique la forme de l'ellipse de confiance et la distribution des données étudiées puisqu'elle prend en compte la variance des données. (Mahalanobis, 1936)

- **Le centroïde**

Le centroïde est le centre des données. Il représente l'intersection entre la moyenne de la première variable x_1 et la moyenne de la deuxième variable x_2. x_1 et x_2 sont les variables de deux métriques étudiées (ex., CPU et TH).

Par définition, le centroïde ou centre géométrique d'une figure plane est la moyenne arithmétique de la position (moyenne) de tous les points de la forme géométrique. Il représente la position moyenne de tous les points dans toutes les directions des coordonnées. (P. C. Mahalanobis)

- **L'ellipse de confiance**

Pour dessiner l'ellipse de confiance, appelée aussi ellipse d'erreur ou de tolérance, il faut tout d'abord calculer les axes de l'ellipse. Cette ellipse permet d'identifier les outliers détectés à l'aide de la distance Mahalanobis. D'après Johnson et Wichern, tout point se trouvant à l'extérieur de cette ellipse est considéré comme étant un outlier. En se basant sur la variance, la plus grande variance sera dans la direction de l'axe des X et la plus petite variance demeure dans la direction de l'axe des Y.

Dans l'ellipse de confiance, nous avons un cas spécial, lorsque l'axe principal de l'ellipse est aligné avec l'axe des X. Dans ce cas, la covariance des données est égale à zéro et les données ne sont pas corrélées.

Prenons l'exemple où les axes de l'ellipse sont alignés avec les axes de coordonnées, l'équation d'une ellipse avec un centre (i, f) et avec les rayons g et h est définie implicitement comme l'ensemble des points (x, y) qui satisfait l'équation suivante :

$$(x-i)\ 2 / g2 + (y-f)\ 2 / h2 = 1. \qquad (3.5)$$

Cependant, pour dessiner l'ellipse, la forme paramétrique est plus utile:

$x(t) = i + g \cos(t)$

$y(t) = f + h \sin(t)$, pour t dans l'intervalle $[0, 2\pi]$ (Cadoret, 2013)

Comme mentionné précédemment , pour calculer une ellipse de confiance , il faut définir les deux axes et l'angle de cette ellipse. L'angle de l'ellipse est déterminé par la covariance des données et ses axes sont calculés en se basant sur les eigenvalues et eigenvectors d'une matrice symétrique, la matrice de variance-covariance $\sum$ dans notre cas.

- Eigenvalue ou la valeur propre de l'ellipse :

Pour la matrice carrée $\sum$ de dimension n*n, nous avons n eigenvalues $\lambda_1, \lambda_2 \ldots \lambda_n$. (Härdle et Simar, 2007) Ces valeurs sont obtenues à l'aide de l'équation suivante :

$$|\sum - \lambda I| = 0 \qquad (3.6)$$

- Eigenvector ou le vecteur propre de l'ellipse :

En général, le vecteur propre $\vec{v}$ d'une matrice symétrique $\sum$ est le vecteur pour lequel on a:

$$\sum \vec{v} = \lambda \vec{v} \rightarrow \vec{v}(\sum - \lambda I) = 0 \qquad (3.7)$$

cela signifie que la transformation linéaire $\sum$ sur le vecteur $\vec{v}$ est complètement définie par λ.

Un eigenvector est un vecteur dont la direction reste inchangée lorsque la transformation linéaire lui est appliquée.

Les eigenvectors correspondants e_1, e_2,…, e_n sont obtenus à l'aide de l'équation suivante : (Härdle et Simar, 2007)

$$(\sum - \Lambda_j I)\, e_j = 0 \qquad (3.8)$$

- **Intervalle de confiance**

Par définition, le niveau de confiance est la fréquence des intervalles de confiance possibles qui contiennent la valeur réelle de leur paramètre correspondant. L'intervalle de confiance traduit la zone d'incertitude relative au résultat en utilisant une méthode d'échantillonnage probabiliste (échantillon aléatoire).
La « largeur » de l'intervalle de confiance est la probabilité souhaitée (généralement 95 %, selon (F. Husson, 2015)) que la valeur étudiée se trouve bien dans cet intervalle.

3.1.2 Approche utilisée

Pour appliquer la méthode de Mahalanobis, nous avons créé trois algorithmes permettant de calculer cette distance :

Algorithme de Mahalanobis

L'algorithme de calcul de la distance Mahalanobis suit les étapes suivantes :

1. Vérifier si les données utilisées suivent une distribution normale. (ligne 3 dans l'algorithme 3.1)
2. Calculer la matrice de variance-covariance et son inverse. (ligne 6 dans l'algorithme 3.1)
3. Calculer la distance Mahalanobis donnée par l'équation 3.1. (ligne 11 dans l'algorithme 3.1)
4. Afficher le graphique montrant la distance mahalanobis de chaque point par rapport au centroid.

Algorithme 3.1 La distance Mahalanobis

```
Input : data1,data2
Output : MD
1.  Function Mahalanobis( data1,data2)
2.  % Vérifier la normalité des données
3.   if ~isnumeric(data1) || ~isreal(data1) || ~isnumeric(data2) || ~isreal(data2)
4.        error('Input x must be a numeric array, x must be positive for log-normality')
5.  %calcul de l'inverse de la matrice de covariance
6.  [ICx,Cx] = mvarcov( data1, data2);
7.  %calcul de la distance mahalanobis
8.  for i=1:length(data1)
9.  val1=data1(i)-mean(data1);
10. val2=data2(i)-mean(data2);
11. MD(i)=sqrt([val1 val2]*ICx*[val1 val2]');
12. end
13. Return MD
14. End
```

Algorithme 3.2 Calcul de la matrice de covariance

```
Input : data1,data2
Output : ICx,Cx
1.  Function mvarcov( data1, data2)
2.  % Calcul de la matrice de covariance
3.  Cx=cov(data1,data2)
4.  ICx=inv(Cx); % l'inverse de la matrice
5.  Return ICx,Cx
6.  End
```

Algorithme de l'ellipse confiance

L'algorithme 3.3 permettant de calculer et de dessiner l'ellipse de confiance est le suivant :

1. Sélectionner les données à traiter. (ligne 3)
2. Calculer la matrice de covariance $\sum$. (ligne 5)
3. Calculer les valeurs propres ($\lambda 1$ et $\lambda 2$) et les vecteurs propres (e1 et e2) de la matrice $\sum$ (voir équations 3.6 et 3.7). (ligne 6)
    a) Ces vecteurs permettent de former les axes de l'ellipse de confiance. Le premier vecteur propre (e1) pointe dans la direction de la plus grande

variance et définit l'axe principal (le grand axe) de l'ellipse. Le deuxième vecteur propre (e2) pointe dans la direction de l'axe mineur (le petit axe).

4. Extraire le plus grand et le plus petit vecteur propre ainsi que les valeurs propres correspondantes. (ligne 8)

5. Calculer l'angle entre l'axe des x et le plus grand vecteur propre. L'angle est entre $-\pi$ et π . (ligne 20)

6. Calculer la valeur critique χ en précisant l'intervalle de confiance I et le degré de liberté p. L'équation 3.2 permet de calculer cette valeur. (ligne 29)

   - I et p peuvent être des vecteurs, des matrices ou des tableaux multidimensionnels de même taille, dépendamment des données traitées. Les paramètres du degré de liberté dans p doivent être positifs et les valeurs dans I doivent appartenir à l'intervalle [0 1].

7. Calculer les axes de l'ellipse (ligne 36 et 37)

   Il est tout aussi simple de paramétrer une ellipse dans les coordonnées définies par les vecteurs propres:

   - Les vecteurs propres permettent de former un cercle à l'aide de la combinaison linéaire décrite dans l'équation 3.9.

$$\cos(t) \times e1 + \sin(t) \times e2 \tag{3.9}$$

   avec t dans l'intervalle $[0, 2\pi]$.

   - L'ellipse est normalisée dans le sens où les écarts types des variables sont utilisés pour suivre la variance des données.Les axes sont calculés comme suit :

$$X\_ellipse = \chi * sqrt(largest\_eigval) * cos(t) \tag{3.10}$$
$$Y\_ellipse = \chi * sqrt(smallest\_eigval) * sin(t) \tag{3.11}$$

   Où largest_eigval est la plus grande valeur propre tandis que le smallest_eigval correspond à la plus petite valeur propre.

8. Dessiner l'ellipse en choisissant l'intervalle de confiance. (ligne 43)

Algorithme 3.3 Représenter l'ellipse de confiance

Input : data1,data2
Output : MD, out_MD, pos_OMD

```
Function mahald(data1,data2)
MD = Mahalanobis(data1,data2)
data = [data1,data2]
% Calculer les vecteurs propres et les valeurs propres
covariance = cov(data)
[evec, eval ] = eig(covariance)
% Spécifier la plus grande valeur propre et le plus grand vecteur propre
[Levec _indice, r] = find(eval== max(max(eval)));
Levec = evec (:, Levec _indice);
Leval = max(max(eval))
% Spécifier la plus petite valeur propre et le plus petit vecteur propre
if(Levec _indice== 1)
   Seval = max(eval (:,2));
   Sevec = evec (:,2);
Else
   Seval = max(eval (:,1));
   Sevec = evec (1,:);
End
% Calculer l'angle entre l'axe des x et le plus grand vecteur propre
Angle_ellipse = atan2(Levec (2), Levec (1));
% l'angle entre -pi et pi
If (angle_ellipse < 0)
    angle_ellipse = angle_ellipse + 2*pi;
End
avg = mean(data);
% Spécifier l'intervalle de confiance 95%
P= [0.95];
for i=1:length(P)
      critic_val =sqrt(chi2inv(P(i),2));
tgrid = linspace(0,2*pi);   %Générer un vecteur linéaire
X=avg(1);
Y=avg(2);
a= critic_val*sqrt(Leval);
b= critic_val*sqrt(Seval);
% Calculer les axes des X et des Y
x_ellipse  = a*cos(tgrid);
y_ellipse  = b*sin(tgrid);
%Definir une matrice de rotation
R=[cos(angle_ellipse) sin(angle_ellipse); -sin(angle_ellipse) cos(angle_ellipse) ];
%Faire la rotation de l'ellipse
r_ellipse = [x_ellipse;y_ellipse]' * R;
```

Algorithme 3.3 Représenter l'ellipse de confiance (Suite)

```
42. % Dessiner l'ellipse de confiance
43. plot(r_ellipse(:,1) + X,r_ellipse(:,2) + Y,'-')
44. % afficher la valeur de la probabilité
45. text(r_ellipse(1,1) + X,r_ellipse(1,2) + Y,[num2str(P(i)*100),'%']);
46. % Plot the eigenvectors
47. quiver(X, Y, Levec (1)*sqrt(Leval), Levec (2)*sqrt(Leval), '-m',
    'LineWidth',2,'ShowArrowHead','off');
48. quiver(X, Y, Sevec (1)*sqrt(Seval), Sevec (2)*sqrt(Seval), '-g',
    'LineWidth',2,'ShowArrowHead','off');
49. end
50. Return MD, out_MD, pos_OMD
51. End
```

3.2 Analyse des performances

3.2.1 Hypothèses

Nous utilisons les mêmes hypothèses mentionnées dans le chapitre 2. Où nous avons considéré :

- Les données utilisées sont des données réelles.
- Les outils de monitorage et de collecte de données sont fiables.

3.2.2 Analyse des résultats

Dans cette partie, nous allons tester la méthode de Mahalanobis avec différentes données à savoir les données d'IMS, les applications cloud de l'ETS et les données de Google. Les scénarios de tests utilisés dans le chapitre 2 sont réutilisés pour comparer les deux méthodes. Nous avons représenté la distance Mahalanobis dans nos résultats en utilisant une barre de couleur. La couleur bleue montre les données ayant une petite distance par rapport au centre des données (connu par son acronyme anglais *« centroid »*). La couleur rouge montre les valeurs extrêmes par rapport au centroid. Ces valeurs représentent les plus grandes distances de Mahalanobis.

Les auteurs dans (F. Husson, 2015) ont montré qu'une probabilité de 95% est suffisante pour calculer la valeur critique et pour illustrer l'ellipse de confiance pour des données avec deux variables. Néanmoins le choix de la probabilité de confiance dépend du problème étudié et de l'objectif à atteindre. Dans notre analyse, nous avons pris un intervalle de confiance de 95%. Pour interpréter les résultats obtenus par la distance Mahalanobis, il faut représenter l'ellipse de confiance. Cette ellipse permet de visualiser et de grouper les données dans un intervalle de confiance.

L'origine de l'ellipse est le centroid. Le premier axe de l'ellipse (représenté en couleur mauve dans toutes les figures représentées ci-dessous) s'étendra le long de l'axe avec la variance la plus grande. Le deuxième axe (représenté en couleur verte sur les figures représentant la distance Mahalanobis) représente l'axe où la variance est la plus faible. Cette ellipse possède deux importantes caractéristiques :

a) Tout point se trouvant sur cette ellipse sera équidistant du centroïde

b) Les axes de l'ellipse sont mis à l'échelle de sorte que les points se trouvant sur la même ellipse soient de la même longueur par rapport au centre. Cette mise à l'échelle a été assurée en calculant les axes et l'angle de l'ellipse à l'aide des eigenvalues et des eigenvectors (voir équations 3.6 et 3.7).

3.2.2.1 Analyse des données IMS

Les figures dans cette section représentent la distance Mahalanobis pour les données d'IMS. Les deux métriques représentées sont le CPU et le TH. Dans un nuage de points appelé aussi diagramme de dispersion, les entrées dans l'axe des abscisses (dans notre cas le CPU) sont représentées par rapport aux entrées correspondantes dans l'axe des ordonnées (dans notre cas le TH). Les scénarios utilisés sont décrits dans le deuxième chapitre de ce travail dans la table 2.2.

3.2.2.1.1 Scénario 1

Pour détecter les outliers, nous avons calculé la distance Mahalanobis afin d'extraire les données bruitées et qui sortent de l'ordinaire dans notre ensemble de données.

La figure 3.2 montre qu'il n'y a pas d'outlier en dehors de notre intervalle de confiance, que nous avons fixée à 95%. Par contre, la valeur de MD indique que le point rouge de coordonnées (45.5,646) peut être considéré comme outlier. La MD pour ce point est égale à 2.2651 qui représente une valeur extrême par rapport à l'ensemble des données. En effet, la distance MD doit être supérieure ou égale à 2.45 pour considérer un point comme outlier (voir équation 3.3).

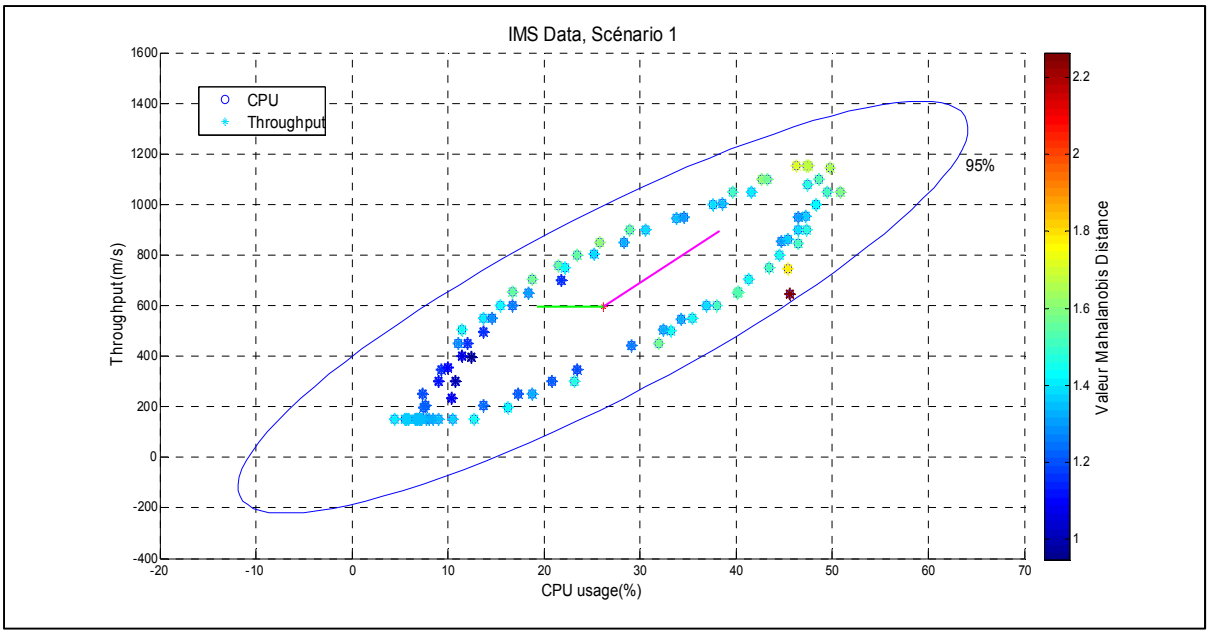

Figure 3.2 Scénario 1 - Approche de détection des outliers basée sur la distance Mahalanobis

Cependant, en analysant les valeurs observées de deux métriques CPU et TH, nous remarquons que ces dernières ne varient pas de la même façon dans temps. Le throughput augmente en premier puis le CPU, c'est pourquoi la méthode de Mahalanobis n'a pas enregistré des outliers puisqu'elle se base sur la corrélation et la correspondance entre les données.

La valeur extrême calculée par le Mahalanobis marqué par une ligne pointillée dans la figure 3.3 (a et b) à la position 73 représente une légère diminution au niveau de TH et une légère augmentation de cpu, ce qui explique sa déviation de l'ensemble des données. Cette variation peut être expliquée par la présence d'une mise à jour effectuée par le système IMS à cet instant.

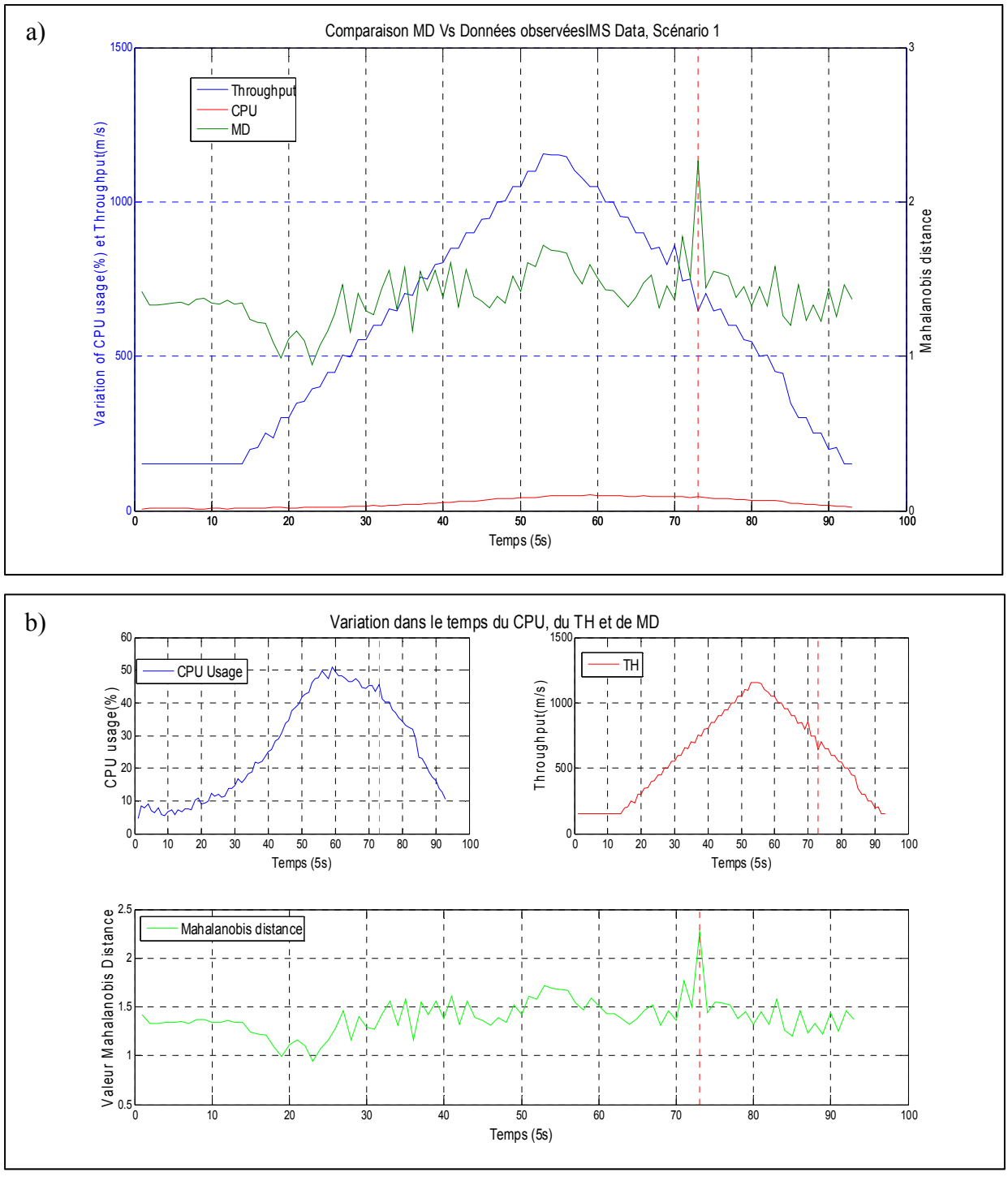

Figure 3.3 Variation dans le temps du CPU, du TH et de MD

3.2.2.1.2 Scénario 2

La figure 3.4 illustre les résultats obtenus pour le scénario 2. Nous avons enregistré six outliers, d'une part puisqu'ils ne font pas partie de l'ellipse de confiance et d'autre part puisqu'ils ont enregistré une grande distance par rapport au centroïde.

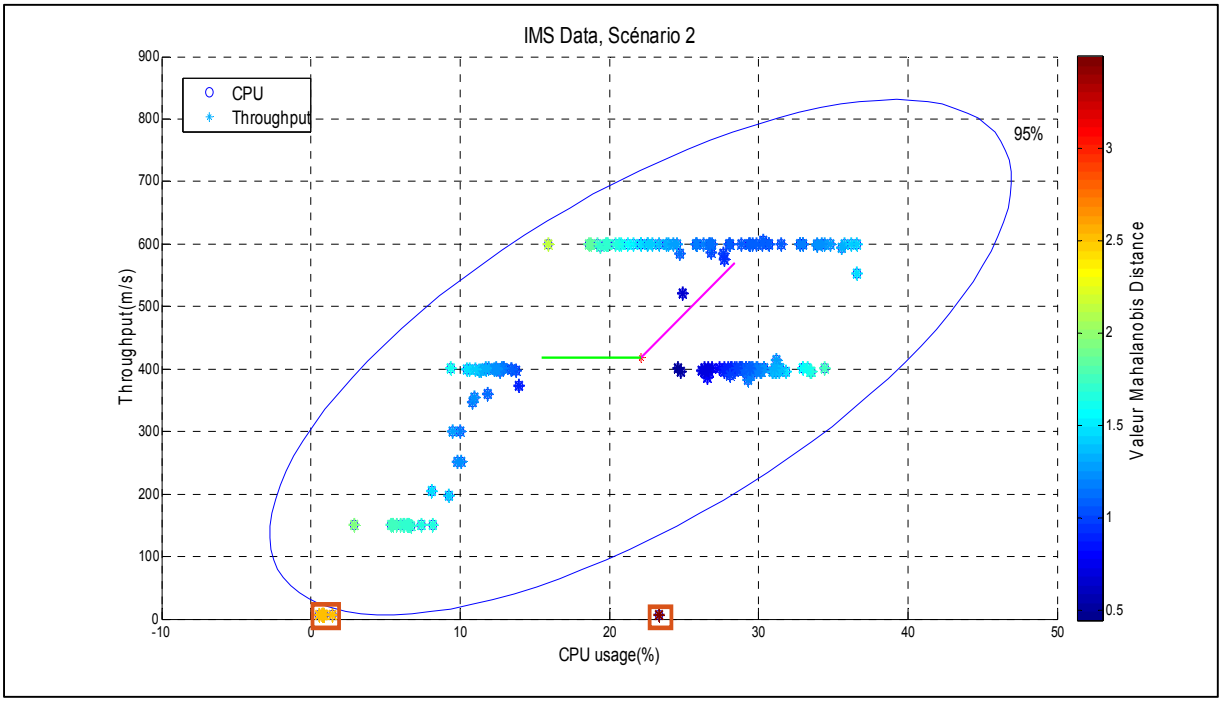

Figure 3.4 Scénario 2 - Approche de détection des outliers basée sur la distance Mahalanobis

Les 6 outliers détectés sont représentés dans le tableau 3.2 en spécifiant leurs positions dans le temps ainsi que la valeur de la distance Mahalanobis.

Tableau 3.2 Valeur de la distance MD des outliers détectés

Position d'outliers	(CPU,TH)	Valeur de MD	Type d'outlier	
			Vrai	Faux
103	(23.33,6)	3.4935	X	
104	(4.4,6)	2.4984	X	
105	(1.6,6)	2.5257	X	
106	(2.2,6)	2.5193	X	
107	(2.4,6)	2.5173	X	
108	(2.6,6)	2.5153	X	

Nous avons enregistré des outliers dans les cas où le throughput est égal à 6 (la valeur minimale de TH). Tous les outliers détectés sont encadrés en rouge dans la figure 3.4.

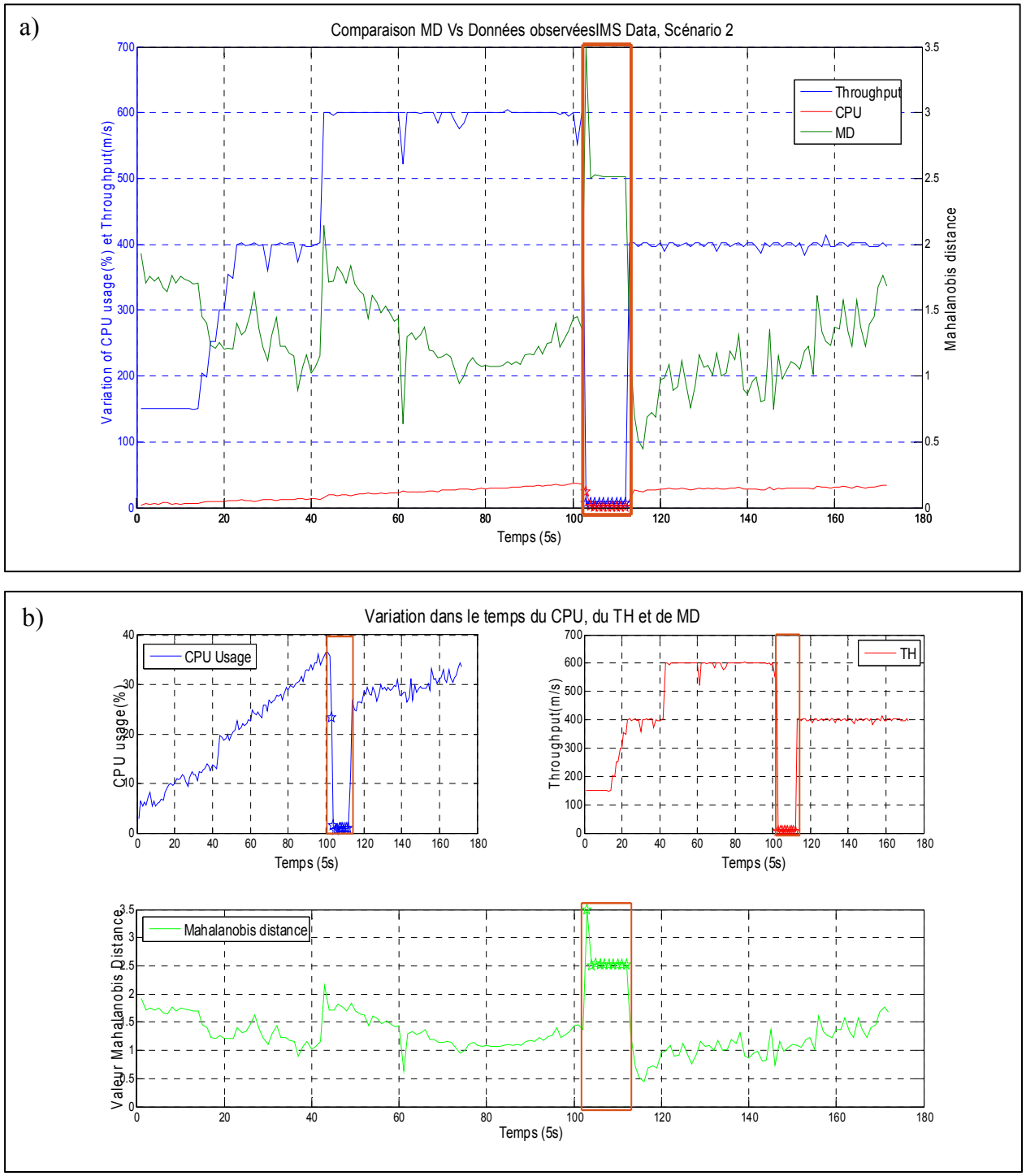

Figure 3.5 Variation dans le temps du CPU, du TH et de MD

Tous les outliers détectés peuvent être considérés comme de vrais outliers. En effet, le premier outlier dans la liste nous indique une chute des valeurs du CPU et du TH, cette diminution est illustrée dans la figure 3.5 (a et b). Les quatre outliers qui suivent nous montrent que le système maintien des valeurs inférieures aux valeurs habituelles (valeurs inhabituelles) et le dernier nous indique une augmentation brusque des valeurs du CPU et du TH. En analysant les données observées des quatre derniers outliers, les valeurs restent inchangées pendant une période de temps de la position 103 jusqu'à la position 108 puis le système s'est rétabli.

Dans la figure 3.5 (b), nous pouvons voir plus clairement la chute de l'utilisation de CPU et de TH dans le même intervalle de temps. On remarque l'augmentation brusque de la valeur de Mahalanobis au début de cette période et puis on voit que ça commencer à devenir plus stable jusqu'à ce que le système reprenne son comportement normal.
En appliquant l'équation 2.3 de fiabilité, nous pouvons conclure que la fiabilité de Mahalanobis pour ce scénario est 100% étant donné que tous les outliers détectés sont des vrais outliers.

3.2.2.1.3 Scénario 3

Dans ce scénario, nous pouvons constater l'existence de 2 outliers : Le premier outlier est le point représenté en rouge dans la figure 3.6 avec les coordonnées (28.8, 198) où la MD est égale à 3.3966 et le deuxième outlier est le point en jaune avec les coordonnées (37.66,600) et où la distance Mahalanobis est de 2.5493.

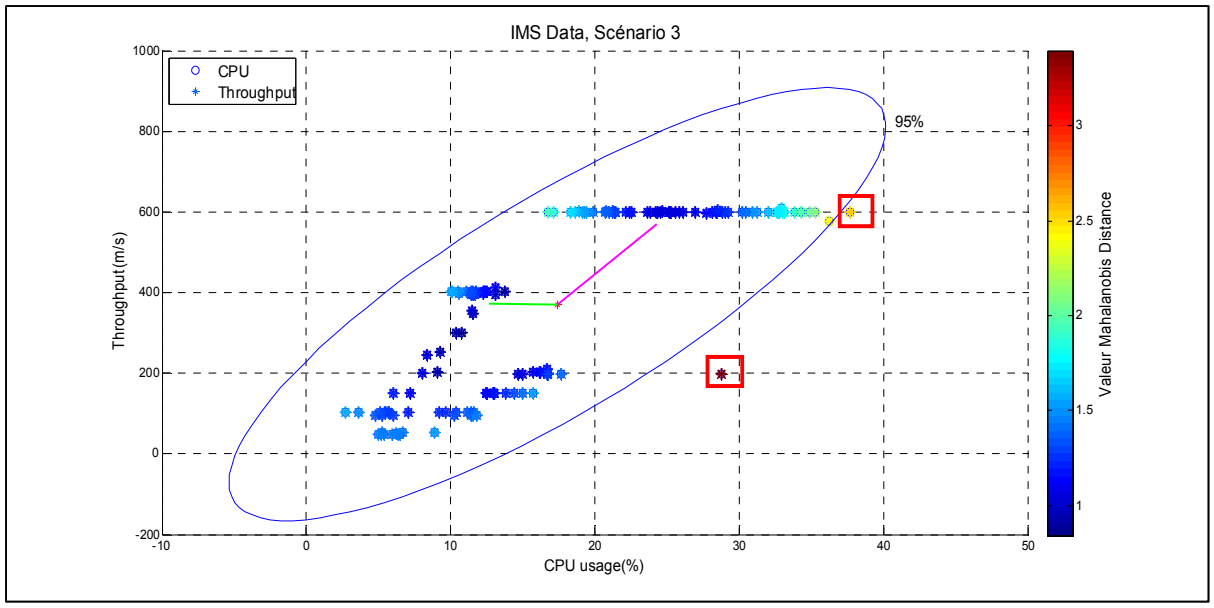

Figure 3.6 Scénario 3 - Approche de détection des outliers basée sur la distance Mahalanobis

Les deux outliers détectés avec la méthode Mahalanobis sont des vrais outliers. Les données observées correspondantes représentées dans la figure 3.7 (a) montrent une augmentation importante en termes de CPU et de throughput au même temps et directement après on enregistre une diminution brusque dans les deux métriques.

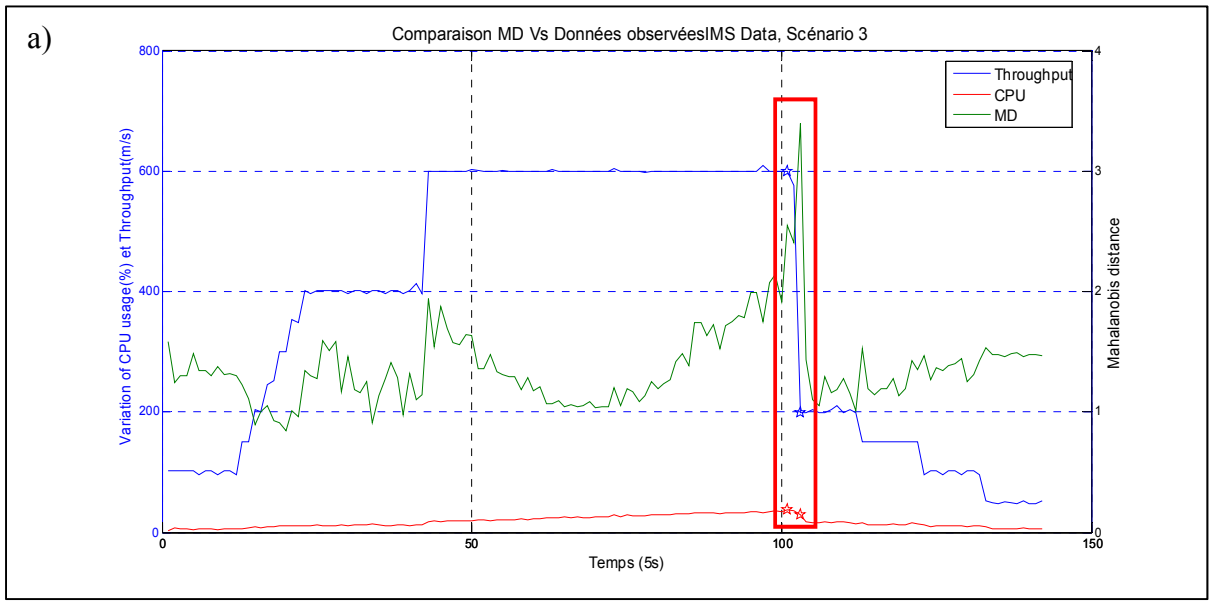

Figure 3.7 (a) Variation dans le temps du CPU, du TH et de MD

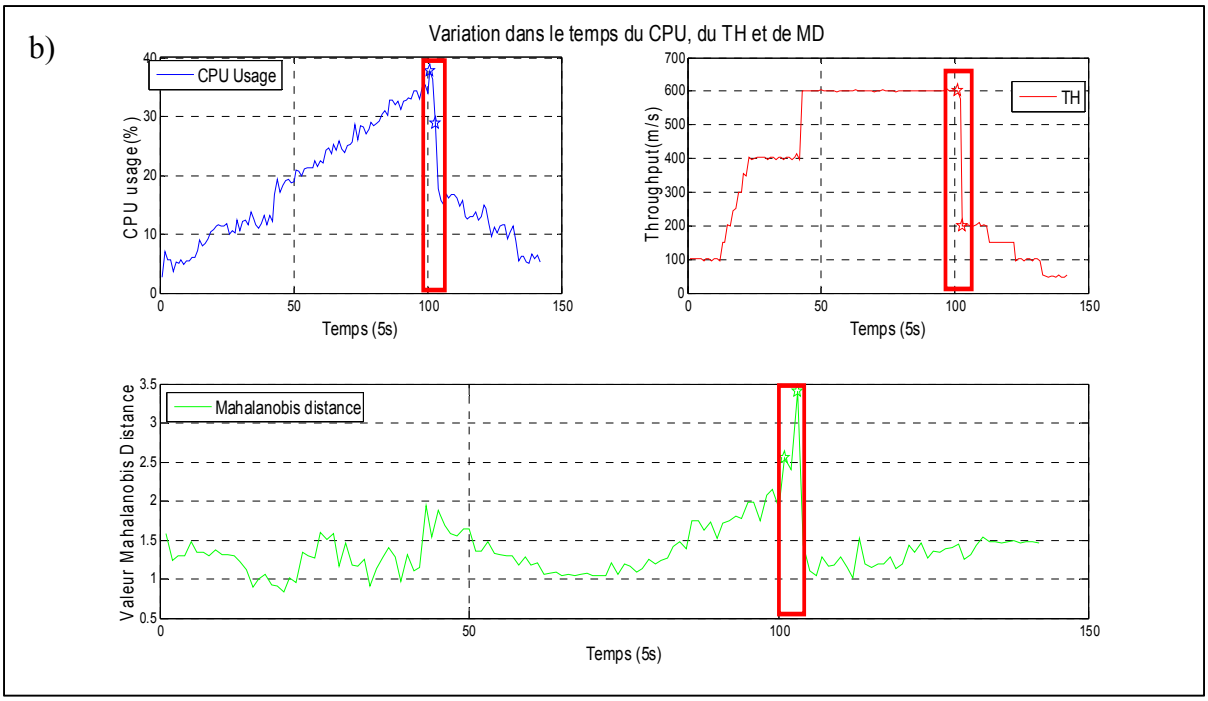

Figure 3.7 (b) Variation dans le temps du CPU, du TH et de MD

La valeur de CPU n'est pas critique, la valeur maximale de CPU pour les deux outliers détectés est 37.66. Cependant la variation des données est importante ce qui explique les résultats obtenus avec la méthode Mahalanobis. Le tableau 3.3 représente les différentes valeurs enregistrées pour les deux métriques.

Tableau 3.3 Valeur de la distance MD des outliers détectés

Position d'outliers	**(CPU,TH)**	**Valeur de MD**	**Type d'outlier**	
			Vrai	**Faux**
101	(28.8,600)	2,5493	X	
103	(37.66,198)	3,3966	X	

La figure 3.7 (b) montre les variations de chaque métrique et de la distance de mahalanobis. On remarque le même comportement pour les deux métriques. Ainsi la distance Mahalanobis n'a connu une augmentation de sa valeur qu'au moment où les deux données varient beaucoup. Les deux valeurs aberrantes détectées sont des vrais outliers. La fiabilité de Mahalanobis pour ce scénario est 100%.

3.2.2.1.4 Scénario 4

Dans ce scénario nous remarquons que la distribution des données diffère des autres scénarios ainsi que la forme de l'ellipse. Dans les scénarios 1 et 2, nous remarquons que le throughput enregistre la plupart du temps une valeur de 600 m/s. Nous pouvons voir que les données dans ce scénario sont différentes et sont plus corrélées.

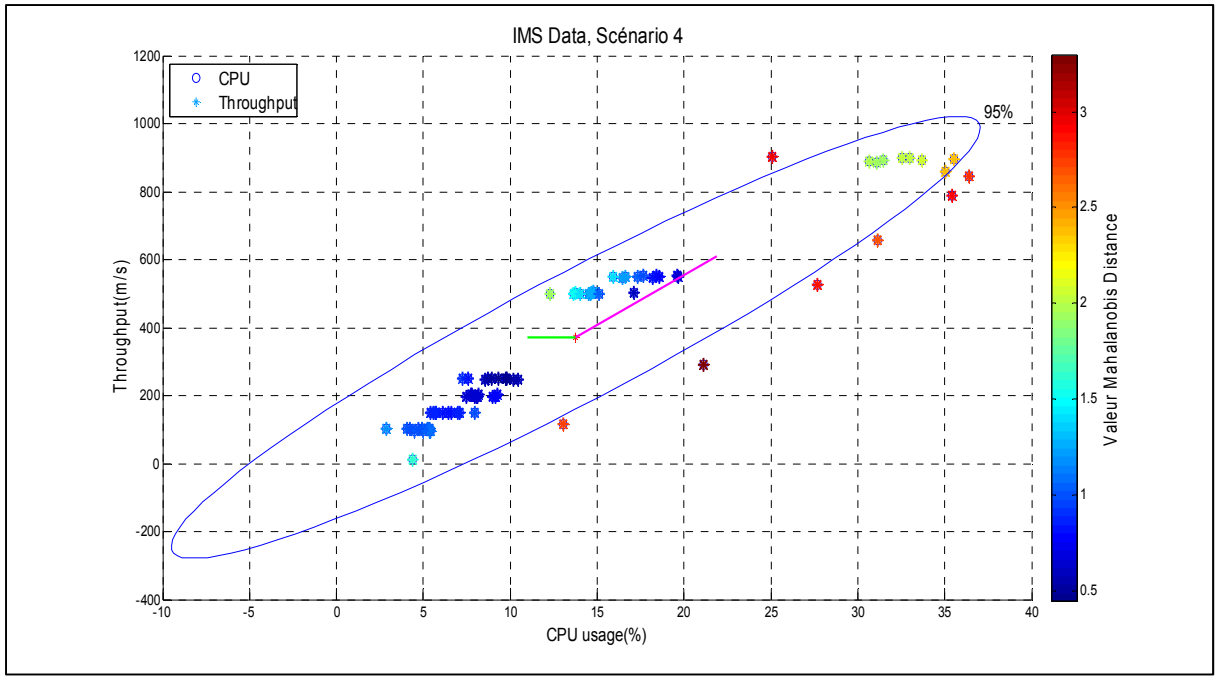

Figure 3.8 Scénario 4 - Approche de détection des outliers basée sur la distance Mahalanobis

À l'aide de l'ellipse de confiance, nous avons pu enregistrer plusieurs outliers qui se situent à l'extérieur de l'ellipse et ont une grande distance de mahalanobis. Dans la figure 3.8, nous remarquons que les outliers détectés sont dispersés dans l'espace.

Dans le tableau 3.4, nous exposons les positions et la valeur de Mahalanobis pour les outliers détectés.

Tableau 3.4 Valeur de la distance MD des outliers détectés

Position d'outliers	(CPU,TH)	Valeur de MD	Type d'outlier	
			Vrai	Faux
63	(25.06,901)	2,9377	X	
72	(36.4,846)	2,7742	X	
73	(35.4,787)	2,9033	X	
74	(31.13,657)	2,7122	X	
75	(27.66,526)	2,8644	X	
76	(21.13,290)	3,2928	X	
77	(13.06,116)	2,7484	X	

Les deux métriques étudiées varient beaucoup dans ce scénario. Les outliers observés apparaissent clairement à travers la variation des données où ils enregistrent une augmentation des ressources puis une diminution brusque. Cette diminution est illustrée dans la figure 3.9 (a). Nous remarquons que pour le CPU et le TH, la variation de données suit la même allure. S'il y a une augmentation de CPU nous constatons une augmentation de Throughput et vice versa.

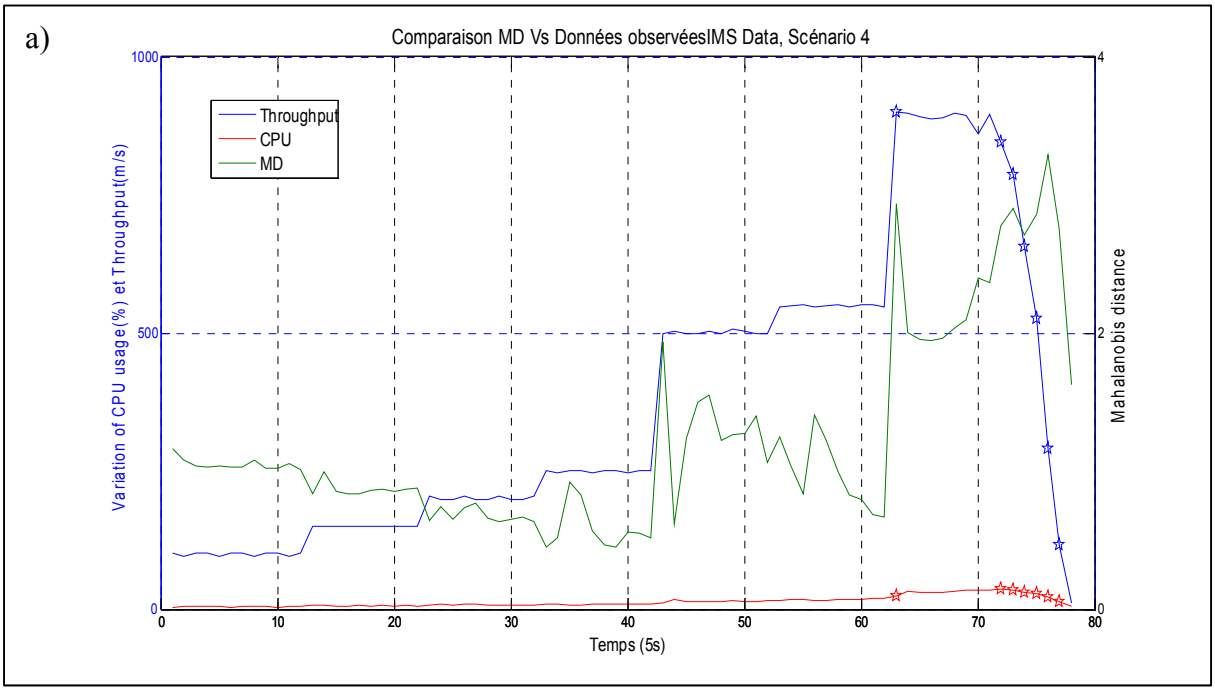

Figure 3.9 (a) Variation dans le temps du CPU, du TH et de MD

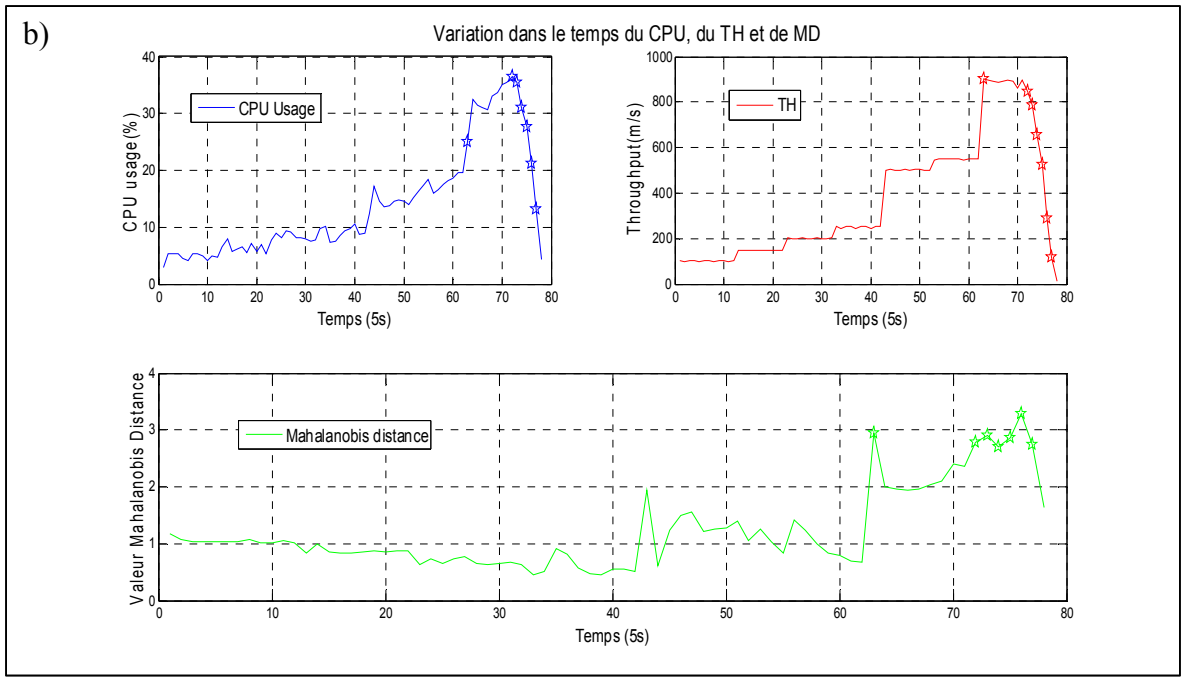

Figure 3.9 (b) Variation dans le temps du CPU, du TH et de MD

La figure 3.9 (b) montre la variation de la distance Mahalanobis ainsi que le CPU et le TH. Nous remarquons la différence entre la variation de l'ensemble de données par rapport au moment de la présence des outliers.On peut conclure que les outliers détectés sont des vrais outliers à partir de cette variation même si la valeur de CPU n'est pas critique. À partir de cette analyse, on peut conclure que tous les outliers détectés sont des vrais outliers. Donc la fiabilité de Mahalanobis est de 100%.

3.2.2.2 Analyse des données de l'ETS

3.2.2.2.1 Scénario 5

Pour le SMTP, nous pouvons remarquer à partir de la figure 3.10 que beaucoup de valeur sort de notre intervalle de confiance de 95%. Ce grand nombre d'outliers est dû aux faites que les données fluctuent beaucoup dans le temps et que les deux métriques varient de la même façon comme montrée dans la figure 3.11 (a). Nous avons enregistré 22 outliers. C'est un grand nombre par rapport aux 5 outliers détectés par la méthode de modified z-score. Cette dernière est basée sur la variation des 3 valeurs ce qui explique la différence en termes des outliers détectés entre ces 2 méthodes. En analysant les données observées, seulement 18 outliers

peuvent être considérés comme des vrais outliers où on trouve que les variations des deux métriques sont importantes et se correspondent dans le temps.

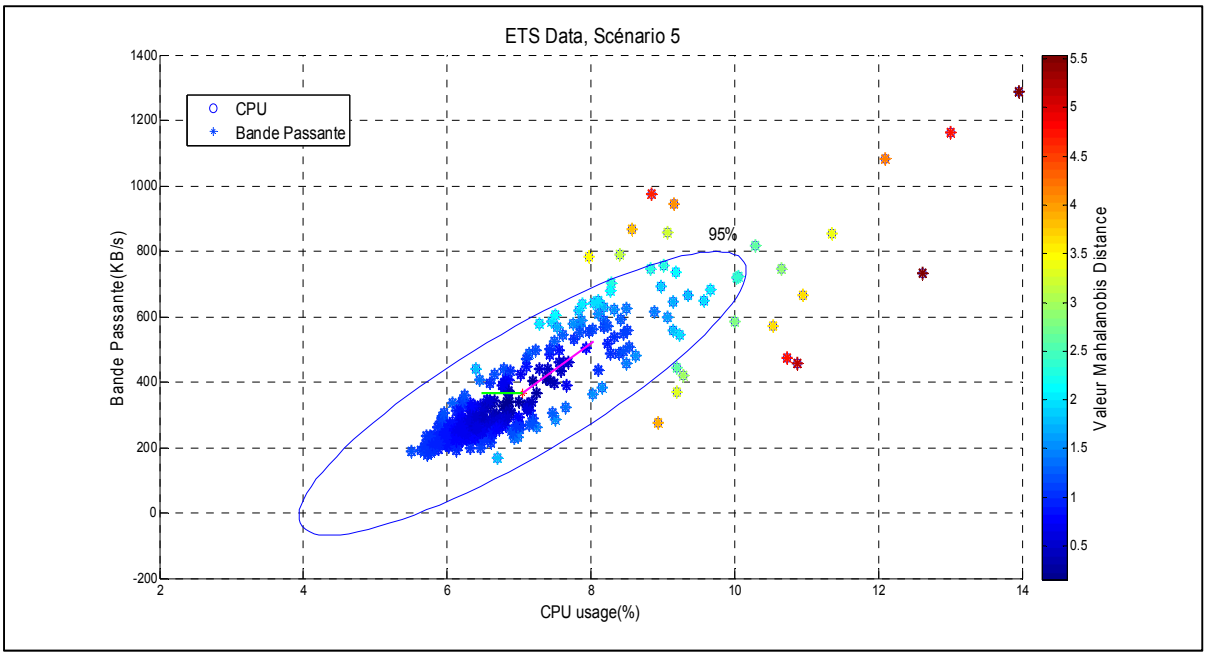

Figure 3.10 Scénario 5 - Approche de détection des outliers basée sur la distance Mahalanobis

Les variations de deux métriques dans le temps montrent des variations répétitives et très variables. Nous pouvons constater des pics qui se répètent dans différents intervalles de temps. Ces variations extrêmes sont surement la source des outliers détectés. La fiabilité de cette méthode pour ce scénario est de 82%.

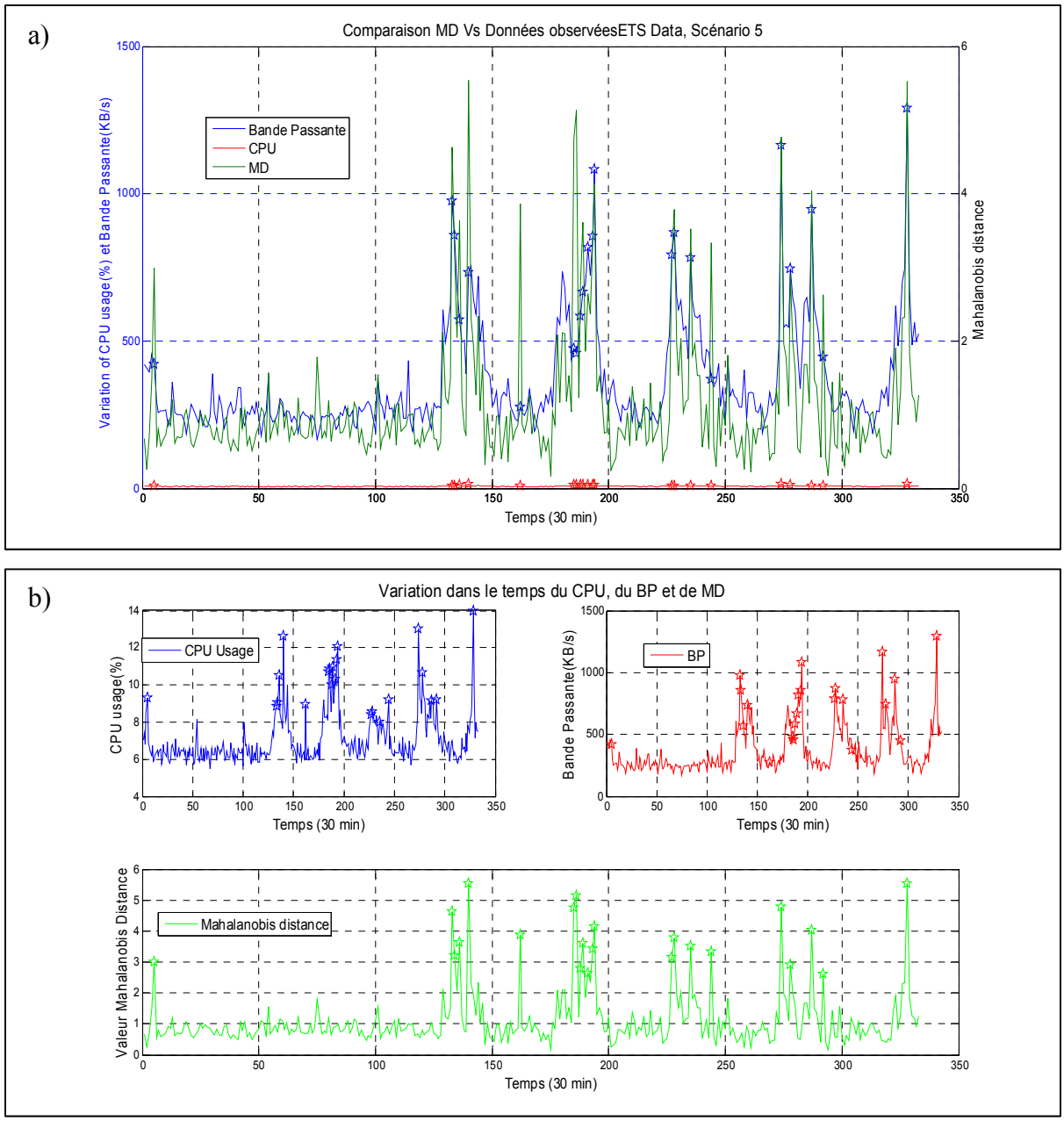

Figure 3.11 Variation dans le temps du CPU, du BP et de MD

3.2.2.2.2 Scénario 6

Dans ce scénario, nous analysons les données web de l'ÉTS. Dans la figure 3.12 on remarque que les valeurs sont un peu dispersées cependant la plupart restent dans l'ellipse de confiance. Plusieurs valeurs ont été identifiées comme des outliers, néanmoins, dans les données

observées on peut voir que dans la même période de temps plusieurs valeurs ont été identifiées comme outlier.

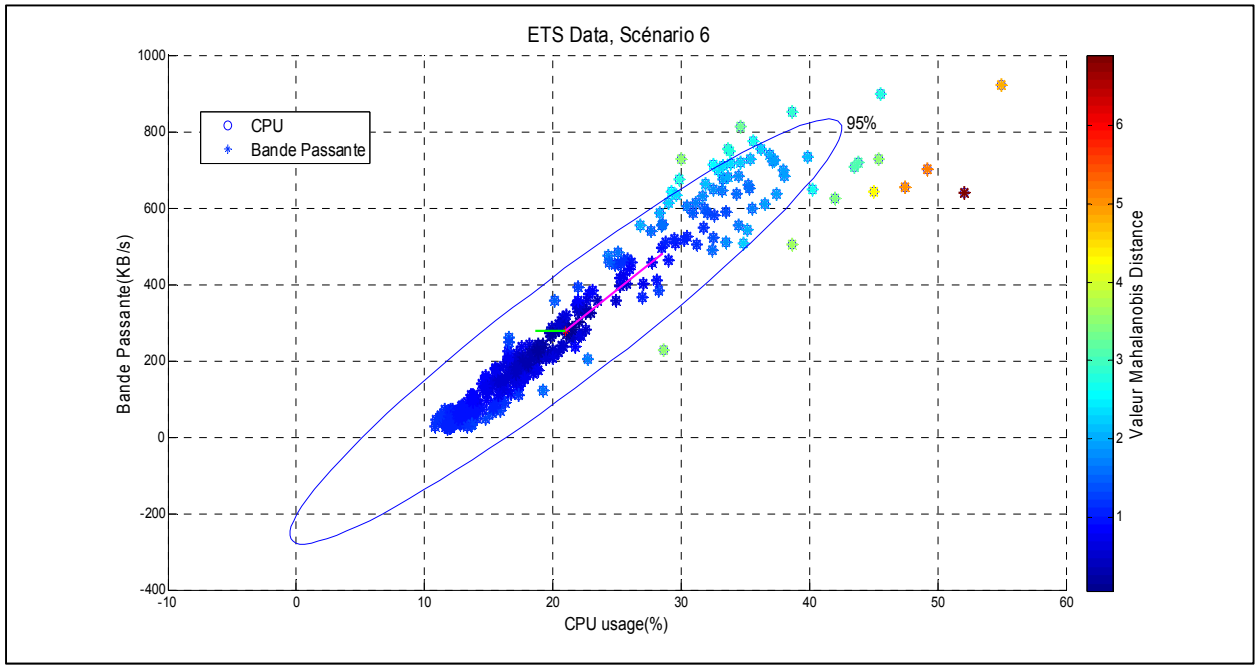

Figure 3.12 Scénario 6 - Approche de détection des outliers basée sur la distance Mahalanobis

On remarque que les données de l'ETS ont un comportement semblable, on trouve des pics à des moments précis puis une chute brusque de l'usage de ressources. Nous ne pouvons pas conclure à l'aide de méthode de détection d'outliers seulement si une observation est une vraie valeur aberrante. L'analyse des données réelles nous a permis de comprendre et comparer les différents comportements de chaque système virtualisé étudié et de calculer la fiabilité de la méthode de détection des outliers.

Avec les données web de l'ÉTS, nous avons identifié 22 outliers. En comparant ces outliers avec leurs données observées on peut garder que 15 valeurs comme des vrais outliers. Les autres sont des faux outliers puisque le système à ce moment n'a pas besoin d'adaptation en termes de ressources. La fiabilité de cette méthode pour ce scénario est de 68%.

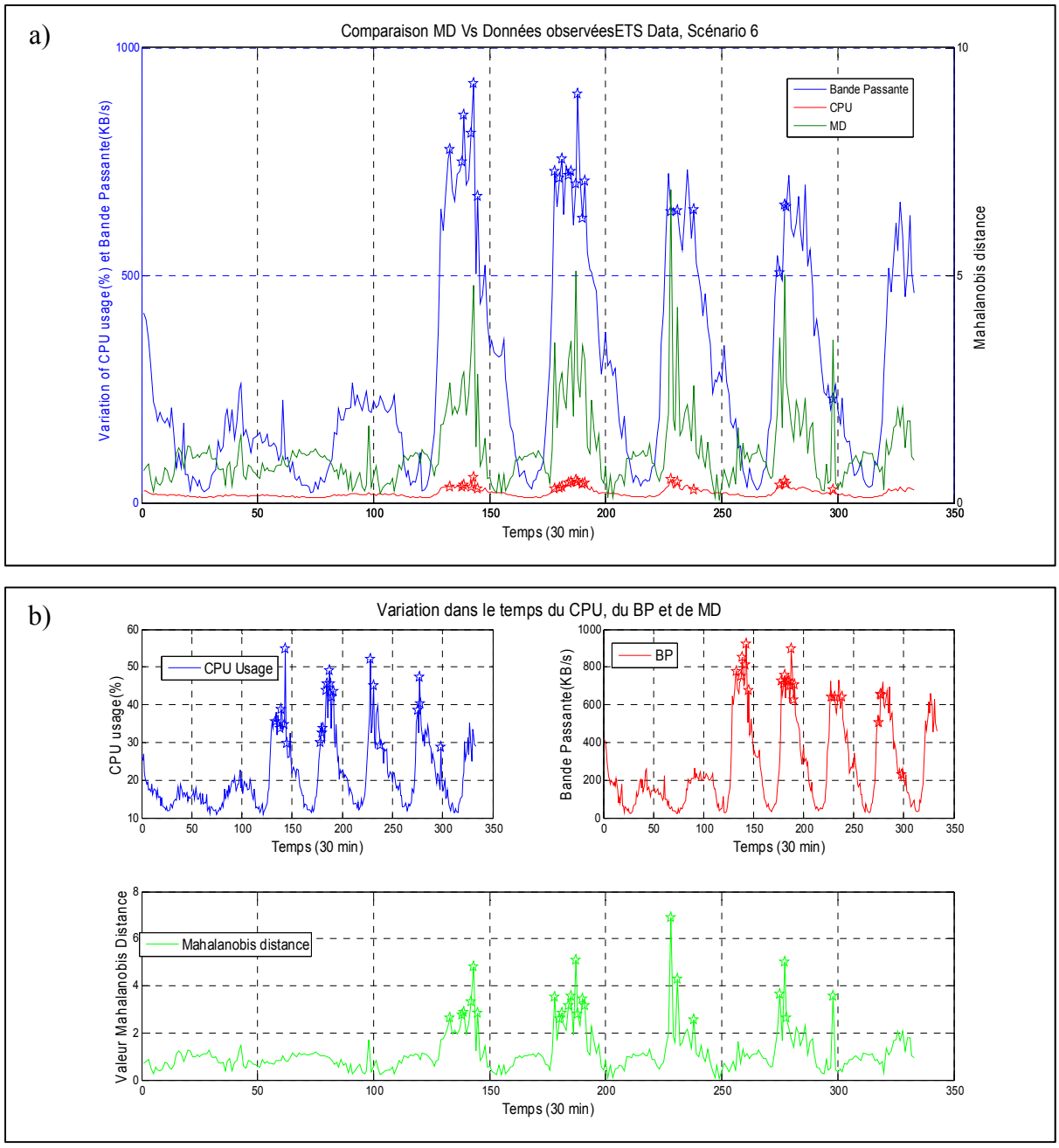

Figure 3.13 Variation dans le temps du CPU, du BP et de MD

3.2.2.2.3 Scénario 7

Ce scénario représente les données SQL de l'ÉTS. Dans ce scénario nous remarquons deux groupes de données qui sortent de notre ellipse de confiance. Pour SQL, le nuage de point est un peu différent. On peut distinguer 3 zones de concentration de données, la première se

retrouve à l'intérieur de l'ellipse de confiance et les deux autres sont à l'extérieur. La variation de MD de ces deux nuages est assez importante, donc dans ce cas, on peut les considérer comme outlier.

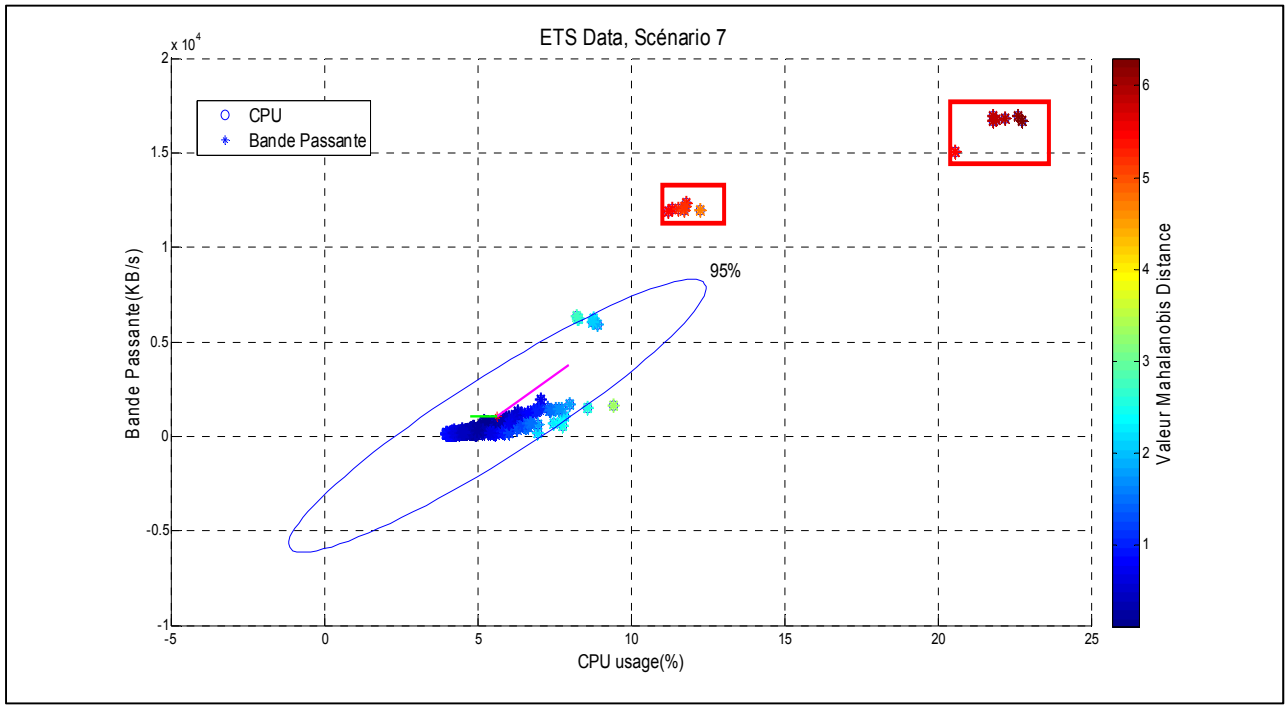

Figure 3.14 Scénario 7 - Approche de détection des outliers basée sur la distance Mahalanobis

En analysant les valeurs réelles de deux métriques, on peut conclure que les outliers détectés représentent vraiment une grande variation en termes de ressources. Ces fluctuations sont à l'origine de ces valeurs aberrantes.

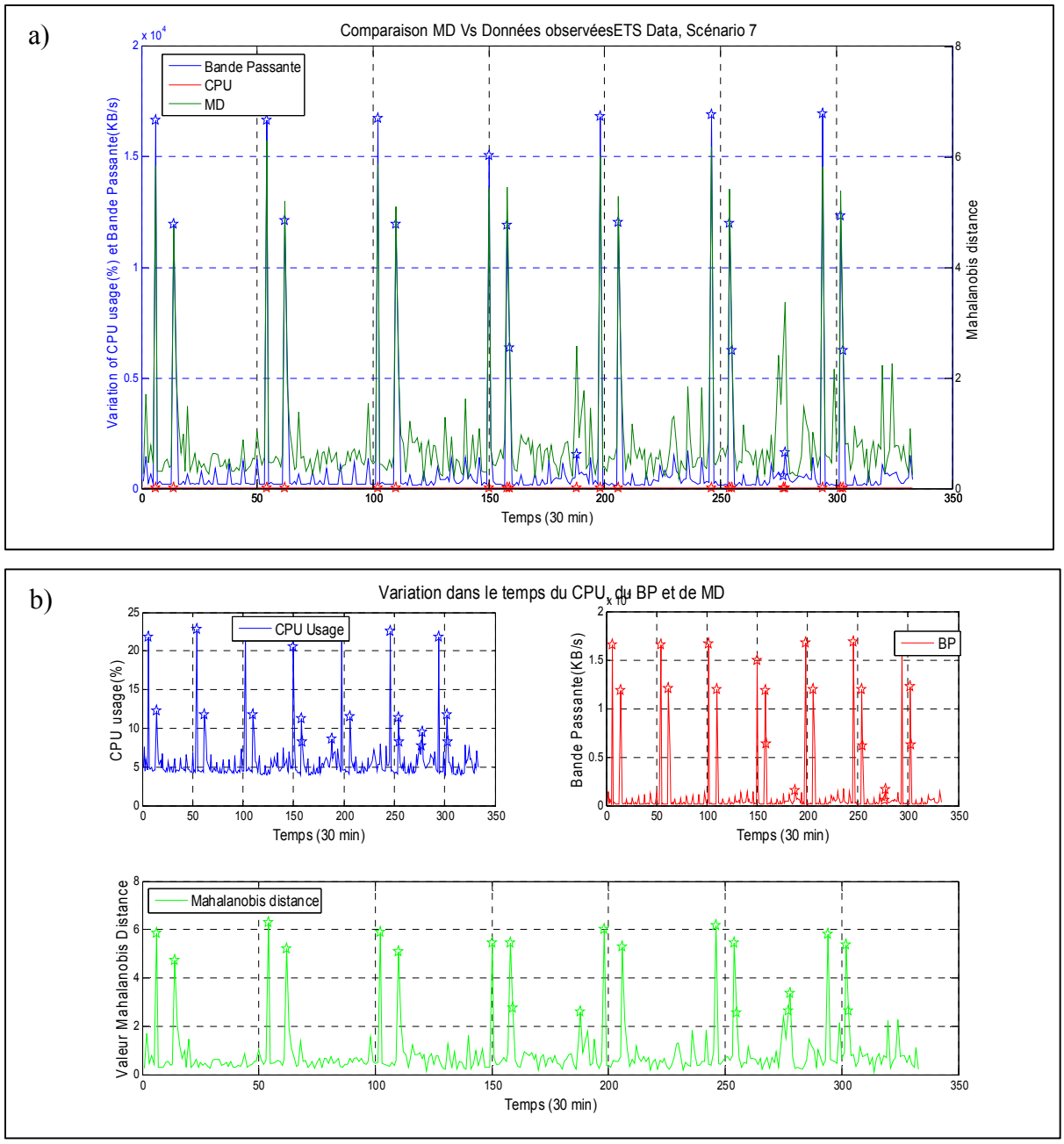

Figure 3.15 Variation dans le temps du CPU, du BP et de MD

Les deux nuages éloignés s'expliquent par la présence des quelques valeurs répétitives. Le premier nuage représente un ensemble des points ayant presque la même valeur de variation tandis que le deuxième nuage représente un autre ensemble de points ayant une valeur plus grande que le premier groupe d'outliers. Le nombre total d'outliers détectés pour ce scénario

est 20 outliers. Pour ce scénario, 16 outliers peuvent être pris en compte. Donc, la fiabilité est de 80%.

3.2.2.3 Analyse des données de Google

3.2.2.3.1 Scénario 8

Nous commençons l'analyse de google avec la tâche numéro 8. Dans ce scénario, nous considérons 10 outliers en total pour l'ensemble de données étudiées. Seulement deux outliers sont des faux outliers. Pour google on considère plus l'usage de CPU et puis on le compare avec la valeur de temps d'exécution de la tâche en question. Cette analyse nous a permis d'écarter deux outliers de l'ensemble détecté.

La fiabilité de cette méthode est de 80%.

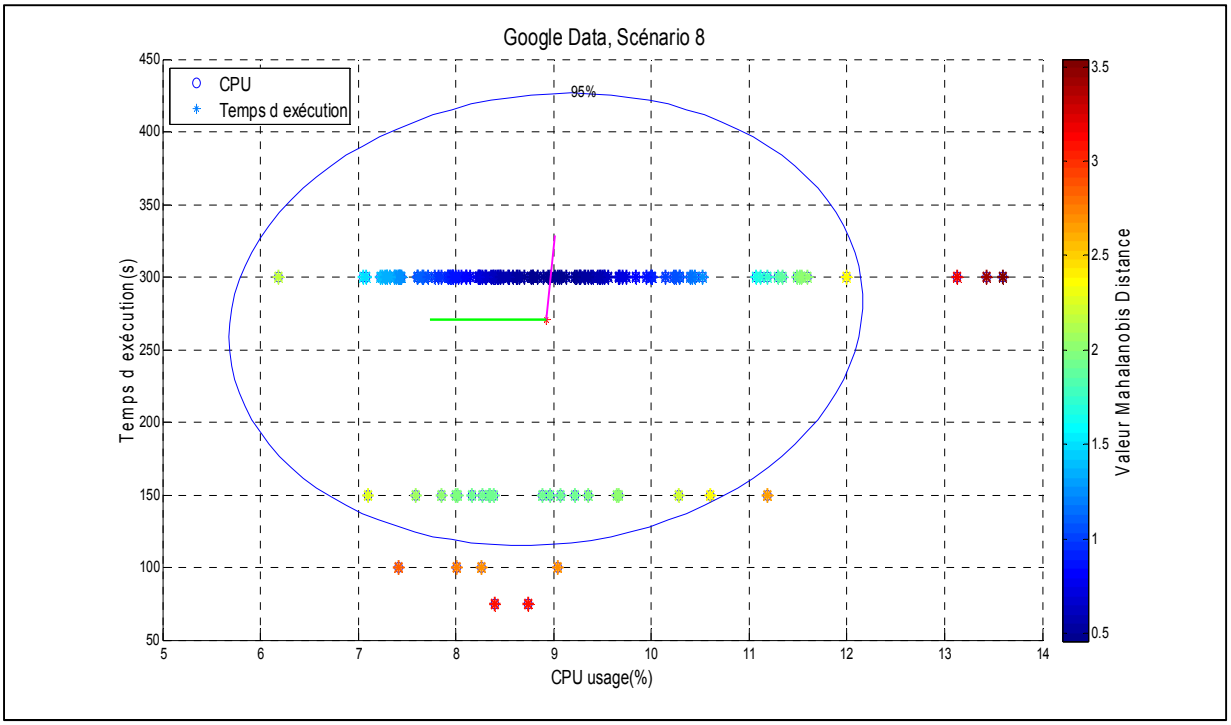

Figure 3.16 Scénario 8 - Approche de détection des outliers basée sur la distance Mahalanobis

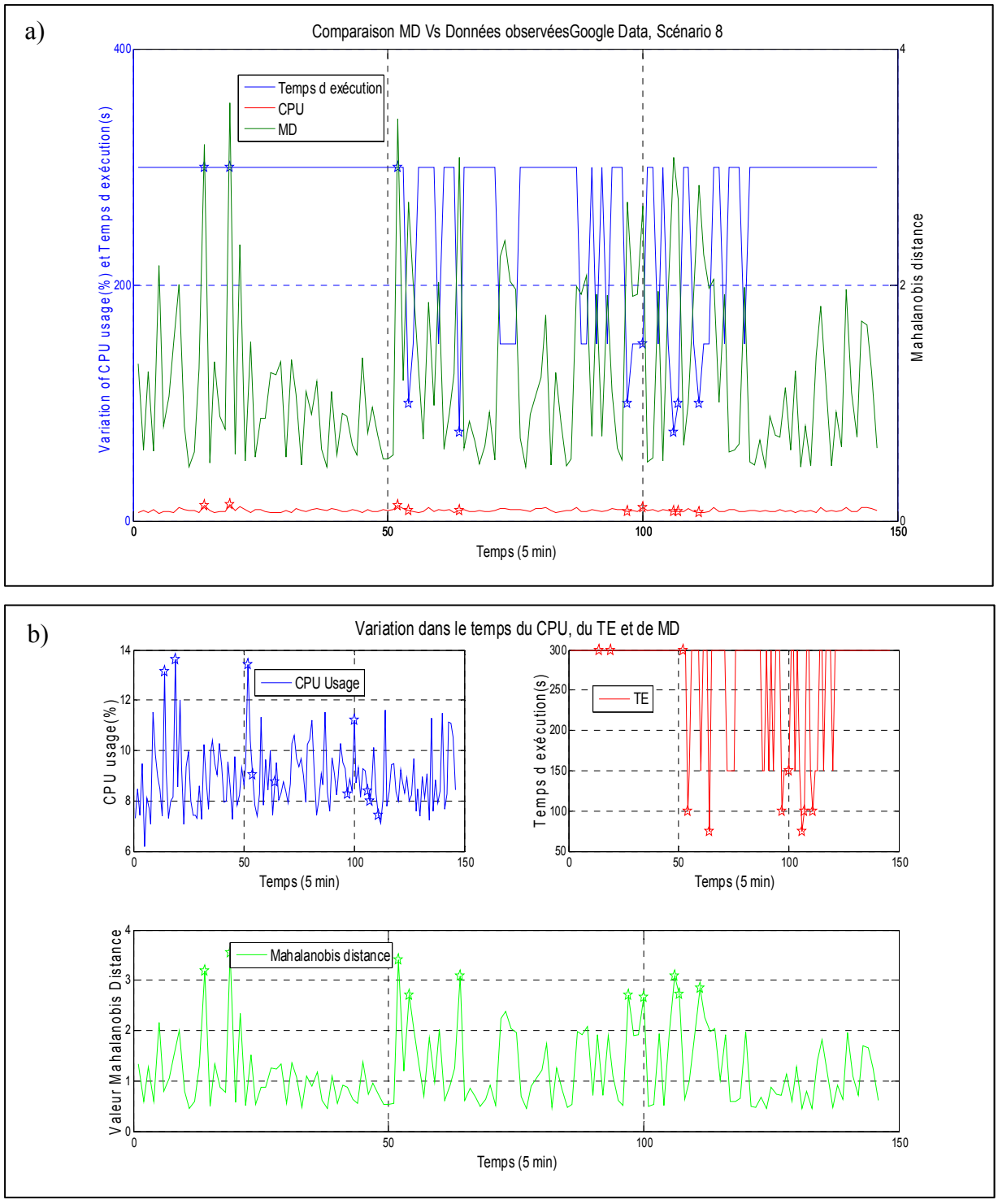

Figure 3.17 Variation dans le temps du CPU, du TE et de MD

3.2.2.3.2 Scénario 9

Le scénario numéro 9 représente la tâche 26 de google où on enregistre 5 outliers. En analysant les résultats obtenus, nous considérons 3 outliers comme des vrais outliers. Donc la fiabilité de Mahalanobis pour ce scénario est de 60%.

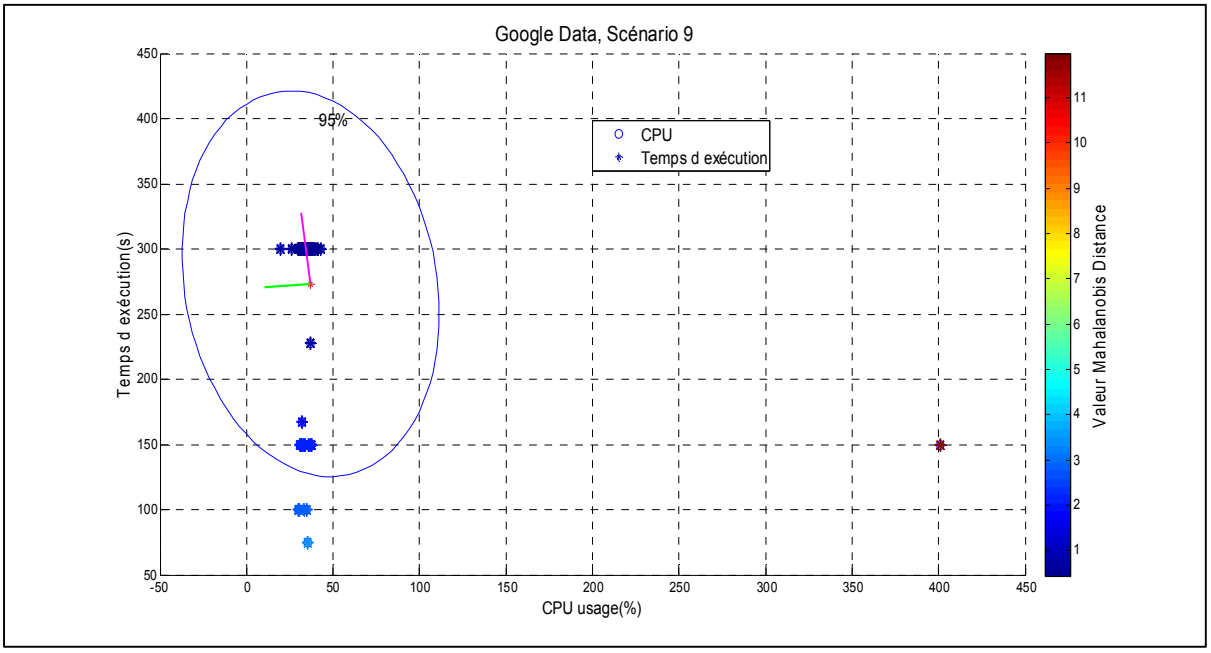

Figure 3.18 Scénario 9 - Approche de détection des outliers basée sur la distance Mahalanobis

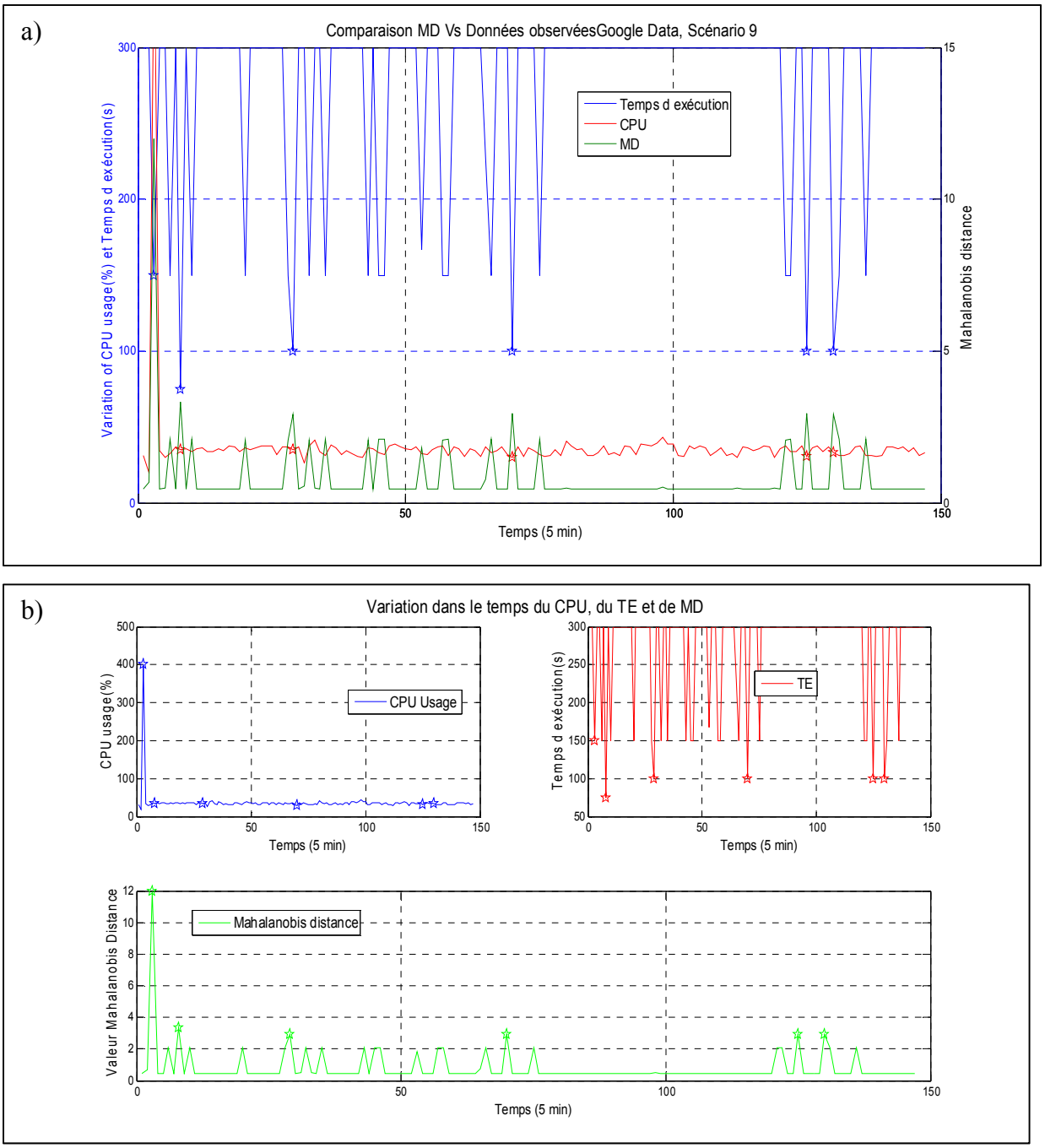

Figure 3.19 Variation dans le temps du CPU, du TE et de MD

3.2.2.3.3 Scénario 10

Dans ce scénario, nous avons pu détecter 7 outliers. Seulement 5 outliers peuvent être considérés. La fiabilité de cette méthode est de 71% (F= 5/7).

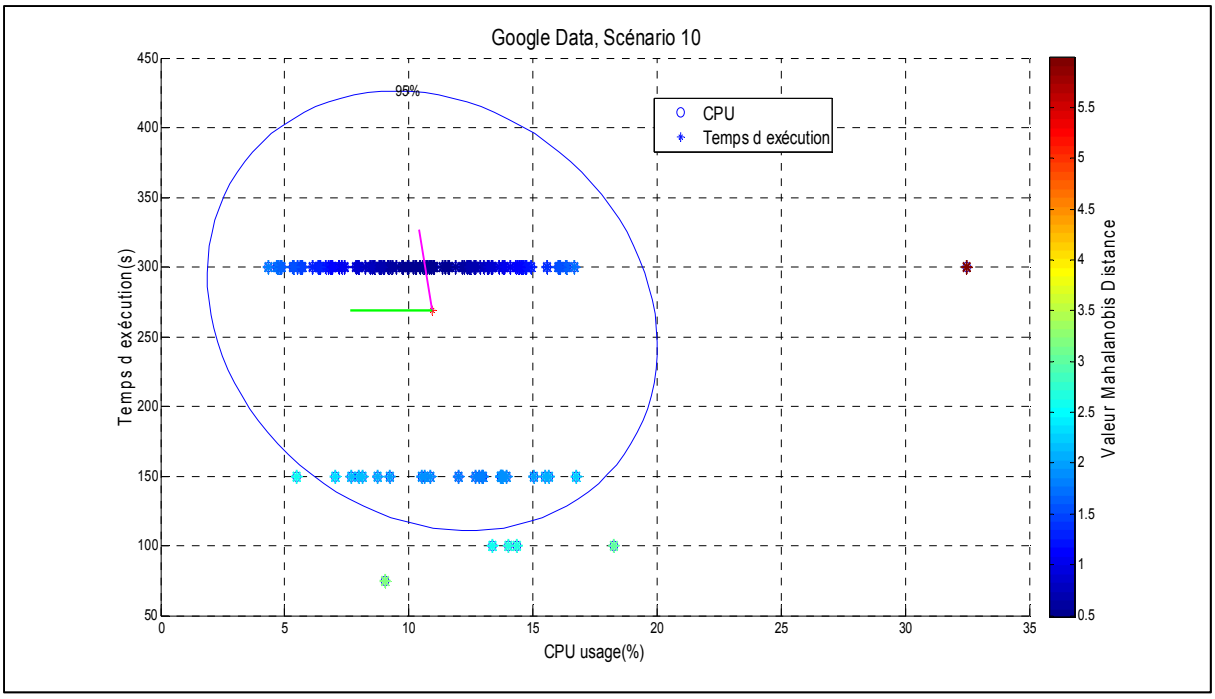

Figure 3.20 Scénario 10 - Approche de détection des outliers basée sur la distance Mahalanobis

a)

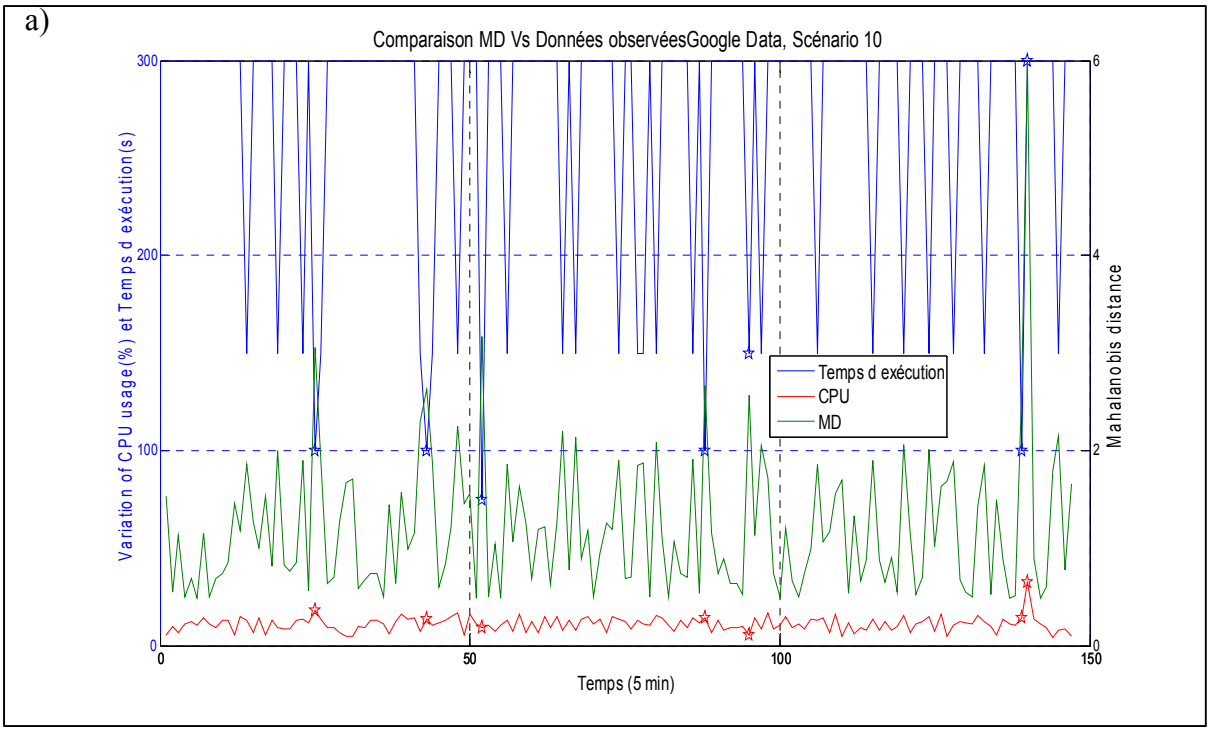

Figure 3.21 (a) Variation dans le temps du CPU, du TE et de MD

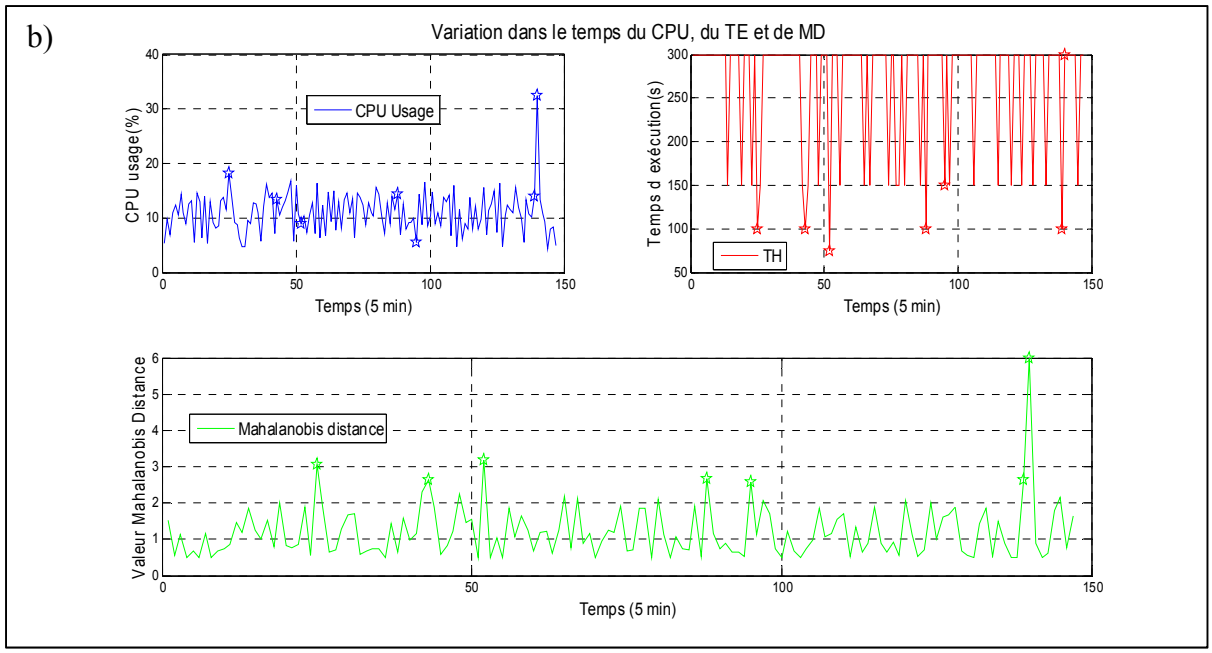

Figure 3.21 (b) Variation dans le temps du CPU, du TE et de MD

3.2.2.3.4 Scénario 11

Nous avons enregistré 20 outliers pour ce scénario de déploiement. La fiabilité de cette méthode est 70% où le nombre des vrais outliers est 14.

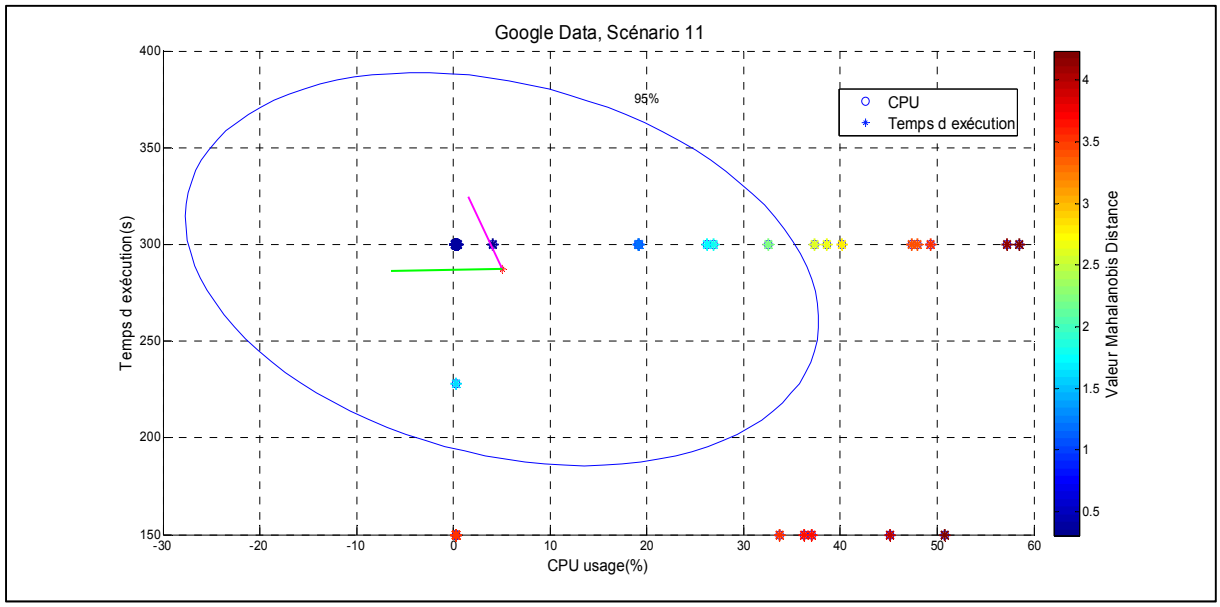

Figure 3.22 Scénario 11 - Approche de détection des outliers basée sur la distance Mahalanobis

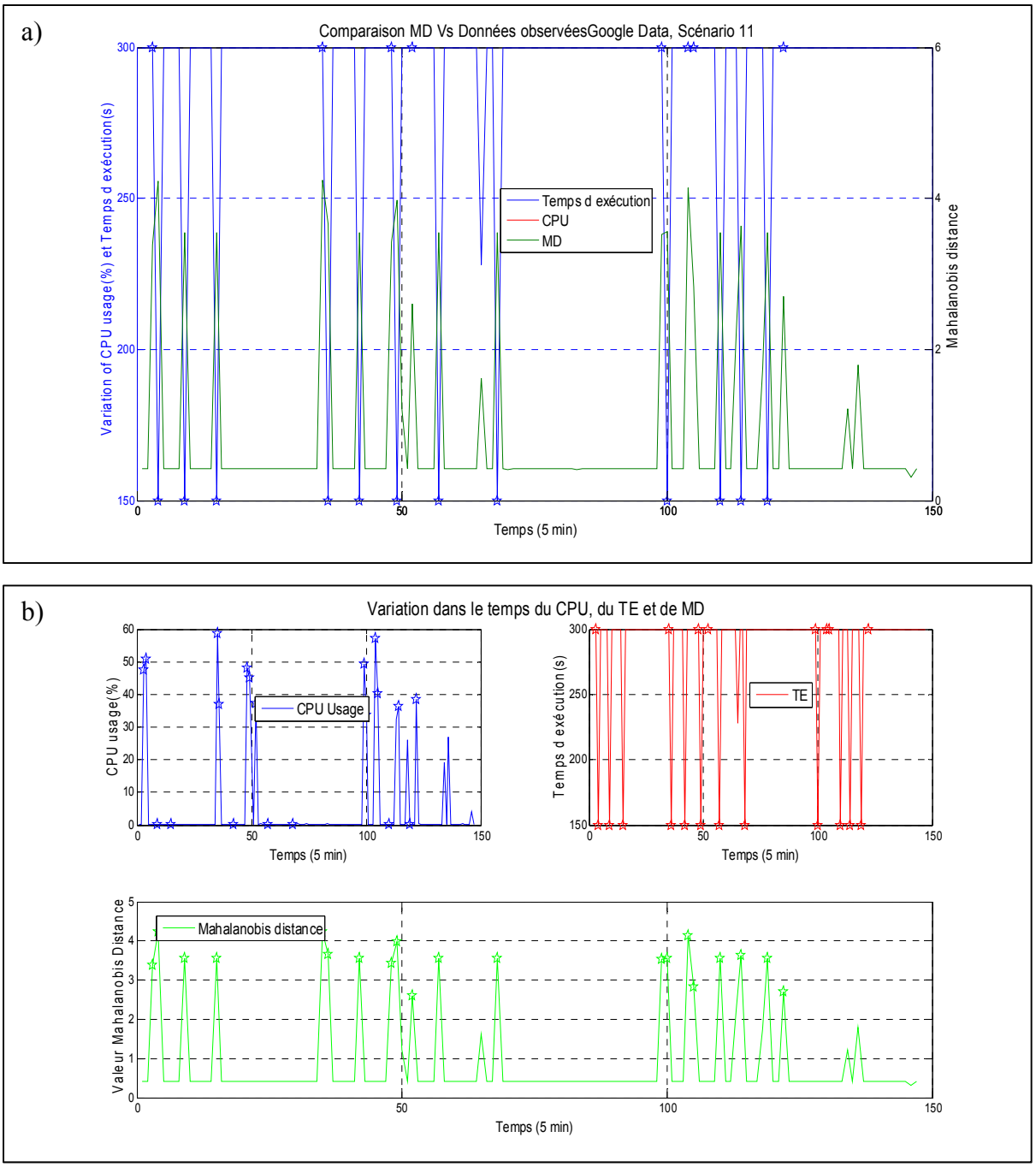

Figure 3.23 Variation dans le temps du CPU, du TE et de MD

3.3 Comparaison entre le Modified z-score et le Mahalanobis distance

Le Modified z-score permet de calculer un score permettant de détecter les valeurs aberrantes pour chaque variable étudiée. Cette analyse permet de comprendre la variation de chaque

valeur. Cependant, le Mahalanobis se base sur la corrélation entre les différentes métriques. Cette dernière nous montre la correspondance entre les données en termes de variation. Cette corrélation est importante pour des systèmes où différentes métriques sont interdépendantes entre-elles (ex. CPU-throughput, Mémoire-délai). Lorsqu'établie, cette interdépendance permet de mieux comprendre le comportement des systèmes et de réagir efficacement face à des changements qui s'opèrent dans ces systèmes (ex. adaptation dynamique des ressources). Dans cette partie, nous allons comparer les résultats obtenus à l'aide de Modified Z-score et ceux résultant de l'approche Mahalanobis.

3.3.1 Données d'IMS

3.3.1.1 Scénario 1

Dans le premier scénario, nous n'avons pas pu détecter aucun outlier avec les deux approches : Modified z-score et Mahalanobis. Cependant, avec l'approche Mahalanobis nous avons une visibilité plus nette sur les données ayant des valeurs extrêmes et qui peuvent représenter un outlier potentiel.

3.3.1.2 Scénario 2

Dans ce scénario de déploiement, seulement 3 outliers ont été détectés en utilisant la méthode Modified z-score comparativement à l'approche Mahalanobis qui en a détecté 6. En comparant ces valeurs avec les variations réelles des données, nous remarquons que le Mahalanobis donne de meilleurs résultats que le Modified Z-score. En effet, au moment de la chute de la valeur du CPU et du TH, le Modified Z-score ne signale pas cette variation au bon moment par contre le Mahalanobis le signale à temps. Avec le Mahalanobis, on détecte les outliers à chaque variation de données tandis qu'avec le Modified Z-score ces variations ne sont pas détectées au bon moment.

3.3.1.3 Scénario 3

Plusieurs outliers ont été détectés dans ce scénario en utilisant le Modified z-score. Seules 2 valeurs ont été considérées comme de vraies valeurs aberrantes en considérant la correspondance entre les 2 métriques étudiées.

En analysant les données réelles, seuls les outliers observés par le Mahalanobis peuvent être des vrais outliers puisqu'ils reflètent deux importants changements dans les données réelles.

3.3.1.4 Scénario 4

En analysant ce scénario, on remarque que le Modified z-score détecte les outliers avant qu'on enregistre la grande variation tandis que le Mahalanobis détecte la variation au moment où il aperçoit un grand changement au niveau des valeurs par rapport à l'ensemble de données. Cette remarque est dûe au faite que le Modified Z-score calcule le score de trois valeurs consécutives tandis que le Mahalanobis calcule la distance pour chaque observation en faisant la correspondance entre les deux métriques étudiées.

3.3.2 Données de l'ÉTS

3.3.2.1 Scénario 5

Pour l'application SMTP, nous avons enregistré 3 vrais outliers parmi les 5 détectés avec le Modified Z-score. Par contre, 22 outliers ont été détectés avec le Mahalanobis dont 18 sont de vrais outliers. Nous remarquons un nombre élevé d'outliers détectés par le MD par rapport au Modified Z-score. Avec la méthode Modified Z-score, on calcule le score pour une fenêtre de trois valeurs à la fois. Donc, si nous avons 3 outliers consécutifs, cette méthode n'enregistre qu'une seule valeur aberrante au lieu éventuellement trois valeurs, en calculant un seul score pour chaque valeur. En plus le Modified Z-score est affecté par les valeurs extrêmes. Il est influencé par l'effet masque qui ne permet pas de détecter tous les outliers présents avec précision.

3.3.2.2 Scénario 6

Pour l'application web, on relève la même observation que pour les autres données. En effet, le Modified Z-score ne détecte pas les outliers au moment où la variation se produit, mais avant que la variation ne soit enregistrée. Le Modified z-score est basé sur le calcul de la variation des valeurs moyennes, cette méthode permet de calculer le score des 3 valeurs observées consécutives. Nous ne pouvons pas déterminer le moment précis où une valeur peut être considérée comme aberrante, mais plutôt un intervalle de trois valeurs qui ont donné ce score.

Ainsi, Modified Z-score permet d'identifier 7 outliers au début de chaque cycle de l'application web, mais cette méthode ne permet pas d'étudier les variations de chaque valeur observée dans ce cycle.

Avec la distance Mahalanobis, on peut voir les variations de chaque cycle d'où les 15 outliers détectés. Elle permet d'identifier les outliers pour lesquels les données sont corrélées et leurs variations sont importantes par rapport à l'ensemble de données.

3.3.2.3 Scénario 7

Avec le modified Z-score, nous avons détecté 14 vrais outliers comparativement à 16 valeurs aberrantes enregistrées par le Mahalanobis. De plus le Mahalanobis, permet d'identifier exactement le moment où la valeur aberrante est détectée.

3.3.3 Données de Google

Pour les données de Google, nous pouvons conclure que l'utilisation de la distance Mahalanobis est plus efficace comparativement à Modified Z-score. Nous n'avons pas pu détecter des outliers avec la Modified Z-score sauf pour le dernier scénario. Cependant, nous n'avons pas pu vérifier sa fiabilité puisque la deuxième métrique étudiée n'a pas été prise en compte et son score n'a pas été calculé. Avec le Mahalanobis nous avons eu de meilleurs résultats. Nous avons obtenu respectivement 8,3,5 et 14 vrais outliers pour les scénarios 8,9,10 et 11. Les principaux résultats obtenus sont résumés dans la sous-section 3.4.4.

3.3.4 Comparaison des méthodes Modified Z-score et Mahalanobis

Le tableau 3.5 récapitule les différentes observations tirées à partir de notre analyse des deux méthodes utilisées à savoir le Modified Z-score et le Mahalanobis. Dans ce tableau, la fiabilité de chaque méthode est listée pour les différents scénarios étudiés.

Tableau 3.5 Comparaison entre les deux approche Modified Z-score et Mahalanobis

Scénarios		Modified z-score	Mahalanobis
IMS	Scénario 1	Pas d'outliers détectés	Pas d'outliers détectés
	Scénario 2	Fiabilité de 67%	Fiabilité de 100%
	Scénario 3	Fiabilité de 50%	Fiabilité de 100%
	Scénario 4	Fiabilité de 75%	Fiabilité de 100%
ETS	Scénario 5	Fiabilité de 60%	Fiabilité de 82%
	Scénario 6	Fiabilité de 86%	Fiabilité de 68%
	Scénario 7	Fiabilité de 78%	Fiabilité de 80%
Google	Scénario 8	-Pas d'outliers détectés -le modified Z-score ne peut pas calculer les scores du temps d'exécution (TE).	Fiabilité de 60% -Le Mahalanobis calcule la distance pour les deux métriques CPU et TE.
	Scénario 9	Pas d'outliers détectés -le TE n'est pas pris en compte	Fiabilité de 60%
	Scénario 10	Pas d'outliers détectés -le TE n'est pas pris en compte	Fiabilité de 71%
	Scénario 11	Fiabilité de 82%	Fiabilité de 70%

Nous remarquons que le Mahalanobis pour les deux scénarios 7 et 11 est moins fiable que le Modified Z-score. La fiabilité du modified Z-score est calculé en tenant compte des outliers que cette approche a détectés. Par exemple pour le scénario 11, d'un totale de 11 outlier ont été identifiés avec une fiabilité de 82%. Cependant, contrairement à l'approche Mahalanobis, cette approche n'a pas détectée tous les outliers présents dans l'ensemble de données. En effet, l'approche Mahalanobis a détecté 20 outliers, dont 14 étaient des vrais outliers avec une fiabilité de 70%.

On peut conclure que même si la fiabilité de Modified Z-score est plus importante pour quelques scénarios, le Mahalanobis reste la méthode la plus fiable des deux puisqu'elle permet non seulement d'identifier tous les outliers, et donc elle n'est pas impactée par l'effet de masquage, mais elle permet aussi de les relevés au bon moment.

Conclusion

Dans ce chapitre, nous avons introduit la méthode de calcul de distance qui est le Mahalanobis. Nous l'avons utilisé pour détecter les outliers dans différents systèmes virtualisés tout en précisant sa fiabilité. Cette analyse nous a permis de faire le choix concernant la méthode à utiliser dans notre mécanisme d'adaptation de ressource qui sera décrit dans le chapitre quatre. Ainsi dans le chapitre suivant, nous allons présenter un algorithme de gestion de ressources dans le cloud en utilisant la méthode Mahalanobis comme méthode de détection des outliers.

CHAPITRE 4

GESTION ET ADAPTATION DES RESSOURCES DANS LE CLOUD

Introduction

Dans ce chapitre, nous allons présenter les motivations qui nous ont poussés à proposer un algorithme d'adaptation des ressources en utilisant les méthodes de détection d'outliers. Nous décrirons l'algorithme proposé ainsi que les résultats obtenus. Nous allons montrer que l'étude faite sur les données à l'aide de l'approche Mahalanobis nous a permis de prédire le moment où les ressources ont besoin d'être réajustées et d'intervenir ainsi au bon moment.

4.1 Motivations

L'objectif principal de la gestion des ressources dans le cloud est l'amélioration de la stabilité des systèmes, qui y sont hébergés, mais aussi la continuité de leurs activités. La surveillance de charge de travail de ces systèmes ne se limite pas à la mesure de métriques (ex., consommation du CPU, bande passante, mémoire) et de statistiques (ex., fréquences de surcharge du CPU). Les industries telles que celle des technologies de l'information et des communications ont des besoins différents en ressources réseau. L'utilisation de ces ressources est très variable dans le temps. Par exemple, pour l'industrie des communications sans fil, la consommation de la bande passante peut présenter des pics importants d'utilisation durant certaines périodes de la journée. Durant ces périodes, il est essentiel de répondre aux besoins des utilisateurs pour éviter la dégradation de la qualité de service. Dans ce cas, les entreprises doivent prévoir à l'avance l'augmentation de la charge de travail imposée aux systèmes afin de fournir les ressources nécessaires au moment opportun (e.g., ajout de cœur CPU, de bande passante, etc.). Cependant, anticiper les besoins en ressources n'est pas toujours évident surtout pour les systèmes qui subissent des fluctuations imprévisibles des demandes des utilisateurs. Il est important d'identifier non seulement la quantité de ressources nécessaires, mais aussi le moment approprié pour les déployer afin d'éviter une surutilisation ou une sous-utilisation prolongée de ces ressources.

Dans le cadre de ce projet, nous proposons une solution qui utilise la méthode de détection des *outliers* qui est le Mahalanobis afin d'identifier les pics d'utilisation et le moment de surcharge des ressources des systèmes. La solution proposée permet d'adapter et d'ajuster les ressources dans les environnements virtualisés afin de garantir une meilleure qualité de service et de minimiser les coûts. Pour évaluer l'approche proposée, nous avons choisi d'étudier à la fois deux métriques, une de haut niveau et une de bas niveau de différents systèmes virtualisés.

Pour les métriques de haut niveau, nous avons décidé de relever les valeurs suivantes :

- Délai d'établissement d'une session média
- Temps de réponse pour une requête
- Nombre de messages

Pour les métriques de bas niveau, nous avons relevé :

- Charge du CPU consommée
- Bande passante consommée

Dans ce qui suit, nous allons décrire l'algorithme implémenté et les résultats obtenus pour les différents scénarios décrits dans le chapitre 2.

4.2 Algorithme d'adaptation des ressources dans le cloud

Dans cette partie nous allons présenter l'algorithme d'adaptation en utilisant le *Modified Z-score* et le *Mahalanobis* et nous allons comparer les résultats obtenus en utilisant les deux méthodes.

Avant de présenter les deux algorithmes des deux approches, nous présentons l'algorithme 4.1 et 4.2 permettant l'extraction et la classification des données à analyser. Les étapes sont les suivantes :

1) Traiter les métriques par ordre de priorité (Algorithme 4.1 ligne 3) :
   Sélectionner les métriques à ajuster en premier : Par exemple, pour chaque type de système nous allons analyser et comparer deux métriques et faire la correspondance entre elles. Cette étape permet de préciser la métrique qui nécessite un ajustement pour garantir la stabilité du système étudié.

2) Ordonner les données (Algorithme 4.1 ligne 5) :

   Représenter les données selon leurs ordres chronologiques dans le temps et préciser une unité de temps entre chaque observation.

3) Spécifier la configuration initiale de chaque type de données (Algorithme 4.1 ligne 7)

4) Extraire les données à analyser (Algorithme 4.1 ligne 8) :

   Créer une variable générique incluant toutes les métriques à étudier pour chaque type d'application.

Algorithme 4.1 Algorithme de chargement de jeu de données

Input : Datapath
Output : Dataset, confIn

1. **Fonction load_dataset (datapath)**
2. Charger les jeux de données
3. Classer les données selon le type d'applications (ex. application de type TI ou télécom)
4. Appel à la fonction **priority**
5. Sélectionner et ordonner les métriques à étudier
6. Enregistrer les métriques à utiliser dans une variable générique englobant toutes les données
7. Enregistrer la configuration initiale du système à analyser dans une variable confIn
8. **Return** Dataset, confIn
9. **End**

Algorithme 4.2 Algorithme de classification de jeu de données

Input : Datapath
Output : prio,data1,datapath,FigTitle,Metric1,Metric2,T

1. **Fonction priority (datapath)**
2. dataset=load_dataset(datapath);
3. %Spécifier le jeu de données à analyser
4. data1 = dataset(:,1);
5. %Spécifier les caractéristiques de chaque jeu de données
6. **if** isequal(data1,dataset_ETS)
7. FigTitle = 'ETS Data';
8. Metric1='CPU usage(%)';
9. Metric2='Bande Passante(KB/s)';
10. T='Temps (30 min)';
11. Prio= Metric1;
12. **elseif** isequal(data1,dataset_IMS)
13. FigTitle = 'IMS Data';

Algorithme 4.2 Algorithme de classification de jeu de données (Suite)

```
    14.  Metric1='CPU usage(%)';
    15.  Metric2='Throughput(m/s)';
    16.  T='Temps (5s)';
    17.  data1 = [dataset_IMS_conf2(:,1)/3
    18.              dataset_IMS_conf2(:,2)];
    19.  prio= Metric1;
    20. elseif isequal(data1,dataset_google)
    21.  FigTitle = 'Google Data';
    22.  Metric1='CPU usage(%)';
    23.  Metric2='Temps d'exécution(s)';
    24.  T='Temps (5 min)';
    25.  prio = Metric1;
    26. End
    27. Return prio,data1,datapath,FigTitle,Metric1,Metric2,T
End
```

4.2.1 Algorithme d'adaptation des ressources basé sur l'approche Modified Z-score

Nous présentons dans cette sous-section, l'algorithme d'adaptation de ressources en se basant sur la méthode de Modified Z-score. Les différentes étapes de cet algorithme sont définies comme suit :

1) Préciser le seuil à utiliser (Algorithme 4.3 ligne 2) :

   Le seuil ayant pour rôle de distinguer les valeurs aberrantes par rapport aux autres valeurs de l'ensemble des données. Cette valeur diffère d'une méthode à une autre. Pour le Modified Z-score, nous devons fixer un seuil qui nous permet de détecter ces variables par contre avec la méthode de calcul de distance Mahalanobis, nous nous fions à un intervalle de confiance permettant de distinguer les différentes données.

2) Extraire les outliers identifiés par la méthode de Modified Z-score pour chaque métrique (Algorithme 4.3 ligne 3) :

   L'algorithme présenté dans le chapitre 2 nous permet de calculer le Modified Z-score[(2)] pour chaque métrique afin d'identifier les valeurs aberrantes (*outliers*).

[2] Méthode présentée dans le chapitre 2

3) Faire la correspondance entre les deux métriques étudiées (Algorithme 4.3 ligne 4) :
   Faire appel à la fonction d'extraction d'outliers (voir Algorithme 2.2 ligne 31)
   Dans cette étape, il faut identifier les outliers pour chaque métrique étudiée (ex., CPU et throughput). S'il y a une correspondance entre deux outliers identifiés, ceci montre que les deux métriques ont le même comportement, soit elles augmentent ou elles diminuent en même temps.
4) Vérifier l'état actuel des ressources du système (Algorithme 4.3 ligne 42) :
   Il faut vérifier la valeur réelle de la métrique par rapport à ce qui lui a été alloué (ex., pourcentage d'utilisation du CPU). Ce ratio nous informe de la capacité d'un système à traiter une charge de travail (ex. un nombre de requêtes plus important) plus élevée afin d'assurer une qualité de service tout en optimisant l'utilisation des ressources (éviter une surutilisation des ressources). Il sera utilisé pour adapter les ressources d'un système soit en ajoutant ou en supprimant des ressources.
5) Comparer la valeur réelle avec la valeur de variation pour chaque outlier (Algorithme 4.3 ligne 4) :
   La variation de chaque métrique est utilisée pour décider si on doit ajouter ou supprimer une ou plusieurs ressources. La comparaison entre la valeur lue de chaque métrique et la valeur de sa variation nous informe si l'outlier détecté est un vrai outlier ou non. Cette information nous indique l'état d'une ressource (ex., sur-utilisée ou sous-utilisée). Par exemple, si la valeur réelle de la métrique est grande, on doit agir soit en ajoutant ou en supprimant des ressources en se basant sur son sens de variation. Ainsi, si la variation est positive, dans ce cas il faut ajouter des ressources, sinon il faut en supprimer.
6) Vérifier et signaler les outliers répétitifs (Algorithme 4.3 ligne 11) :
   Nous devons calculer la fréquence d'apparition d'un outlier. Cette fréquence représente le nombre de fois un outlier est identifié et l'intervalle de temps durant lequel il apparait. Si plusieurs outliers se répètent durant un intervalle de temps réduit, à ce moment, il faut agir, en procédant selon les étapes décrites précédemment.

7) Signaler l'existence d'un vrai outlier (Algorithme 4.3 ligne 27) :

   Pour signaler la présence d'un vrai outlier, il faut identifier si un outlier est un vrai outlier ou non. Si l'outlier est signalé comme étant vrai, une décision est prise pour ajouter ou supprimer des ressources selon le sens de variation des métriques.

Cette décision est prise à l'aide d'une variable *Flag,* une variable booléenne qui prend les valeurs 0, 1 ou -1. Elle permet de nous informer si la ressource étudiée a besoin d'un ajustement ou non (0 ou 1/-1). En cas d'ajustement, la valeur 1 indique qu'il faut ajouter des ressources alors que la valeur -1 indique qu'il faut supprimer des ressources.

Les règles de décision :

Pour la prise de décision, il faut considérer plusieurs paramètres :

1) Type d'application : dans notre cas, nous avons étudié les données IMS et d'une application Web de l'ÉTS. (Voir algorithme 4.3 ligne 1)
2) Le sens de variation des outliers identifiés : le sens nous permet de prendre une décision quant à l'ajout ou à la suppression des ressources. (Voir algorithme 4.3 ligne 2)
3) La disponibilité des ressources : Savoir les ressources disponibles pour chaque système (ex., nombre de cœurs CPU et bande passante disponibles,). (Voir algorithme 4.3 ligne 1)
4) L'état actuel des ressources dans le système : vérifier le taux d'utilisation des ressources et le signaler. Par exemple, si les taux d'utilisation du CPU et de la bande passante atteignent des seuils de 80% par rapport aux capacités maximales disponibles, l'algorithme doit le signaler. (Voir algorithme 4.3 ligne 42)

Algorithme 4.3 Adaptation de ressources en utilisant le Modified Z-score

Input : Prio, data1,datapath, confIn
Output : Flag , Adapt

1. **Fonction adapt_mzscore (Prio, data1,datapath, confIn)**
2. Appel à la fonction de Modified Z-score
3. Extraire les outliers avec la fonction de Modified Z-score
4. **% Comparaison de la variation et la valeur observée pour chaque métrique**

Algorithme 4.3 Adaptation de ressources en utilisant le Modified Z-score (Suite)

```
5.  For c=1:length(Out_pairs)
6.      valR=[meandata(Out_pairs(c),3) meandata(Out_pairs(c),4)];
7.      varmax=vardata(Out_pairs(c),:); %valeur de la variation
8.      matrix_M1(c,:)=[valR(1,1) varmax(1,1)];
9.      matrix_M2(c,:)=[valR(1,2) varmax(1,2)];
10. End
11. %Identifier les outliers répétitifs dans les deux métriques
12. t=1;y=1;
13. while t<length(O_metric1) % première métrique
14. rep1(t)=O_metric1(t+1,1)-O_metric1(t,1);
15. if rep1(t)==1
16.         out_rep1=[O_metric1(t,1);O_metric1(t+1,1)];
17. End
18.  t=t+1;
19.  End
20.  while y<length(O_metric2) % deuxième métrique
21.       rep2(y)=O_metric2(y+1,1)-O_metric2(y,1);
22.       if rep2(y)==1
23.          out_rep2=[O_metric2(y,1);O_metric2(y+1,1)];
24.     End
25.  y=y+1;
26.  End
27. % Signaler le besoin d'un ajustement
28. h=1;c=1;
29. while h<=length(O_pairs)
30. if strcmp(prio,'CPU')
31.      if (matrix_M1(h,1)>50) && (matrix_M1(h,2)>=0)  %CPU>50 et max positif
32.       flag(c,:)=[1 O_pairs(h)];
33.       elseif (matrix_M1(h,1)>50) && (matrix_M1(h,2)<0) %CPU>50 et max
    variation negatif
34.       flag(c,:)=[-1 O_pairs(h)];
35.     Else
36.          flag(c,:)=[0 O_pairs(h)];
37.       End
38.              c=c+1;
39.        end
40.   h=h+1;
41. Endwhile
42. % Valeur actuelle de la ressource
43. for c=1:length(vardata)
44.         var=vardata(c);
45.         cpu_act=(maxCPU(c)/ confIn(:,1))+var; %Le max de cpu d'un seul core
46.         if(cpu_act>50)
```

Algorithme 4.3 Adaptation de ressources en utilisant le Modified Z-score (Suite)

```
47.          v_act(c,:)=[cpu_act,1];
48. Else
49.          v_act(c,:)=[cpu_act,0];
50.       End
51. End
52. % Faire la décision d'ajustement des ressources
53. For i=1 :legnth(O-pairs)
54.   if flag =1 && v_act =1
55.         Adapt = 1 % Ajouter 1 cœur
56.         confIn(:,1) = confIn(:,1)+1 %Actualiser le nombre des cœurs disponibles
57.         confIn(:,2) = confIn(:,2)*2 %doubler la capacité de la 2ème métrique
58.   else if flag =1 && v_act =0
59.         Adapt  = 0
60.   elseif flag =-1 && v_act =1
61.         Adapt =-1 % Supprimer 1coeur
62.         confIn(:,1)= confIn(:,1)-1 %Actualiser le nombre des cœurs disponibles
63.         confIn(:,2)= confIn(:,2)/2 %doubler la capacité de la 2ème métrique
64.    Else
65.         Adapt =0
66. End
67. Return Adapt, flag
68. End
```

4.2.2 Algorithme d'adaptation des ressources basé sur l'approche Mahalanobis

Dans cette section, nous allons présenter l'algorithme d'adaptation de ressources basé sur la méthode de calcul de distance Mahalanobis. Les mêmes étapes présentées à la section 4.2.1 ont été suivies pour cette méthode à l'exception de l'étape 3. Cette dernière a servi à établir la correspondance entre les outliers détectés au niveau des deux métriques pour le Modified Z-score. L'algorithme 4.4 décrit les étapes de calcul de l'approche d'adaptation basée sur le Mahalanobis.

Algorithme 4.4 Adaptation de ressources en utilisant le Mahalanobis

```
Input : Prio, data1,datapath, confIn
Output : decision,confIn
   1. Fonction mahaladapt
   2. %Appel à la fonction priority
```

Algorithme 4.4 Adaptation de ressources en utilisant le Mahalanobis (Suite)

```
3.  [prio , data1, datapath, confIn] = priority
4.  %Appel à la fonction mahald
5.      [ MD, out_MD, pos_OMD ] = mahald(data1(:,1),data1(:,2));
6.  %Identifier les outliers répétitifs
7.  t=1;y=1;
8.  while t<length(out_MD)
9.       rep1(t)=pos_OMD(t+1,1)-pos_OMD(t,1);
10.      if rep1(t)==1
11.         out_rep1(y,:)=[pos_OMD(t,1);pos_OMD(t+1,1)];
12.         y=y+1;
13.         R(t)=1;
14.      Else
15.          R(t)=0;
16.      End
17.      t=t+1;
18.  End
19. % Extraire les valeurs réelles des outliers pour les 2 métriques
20.    for c=1:length(pos_OMD)
21.       valR(c,:)=[data1(pos_OMD(c),1) data1(pos_OMD(c),2)];
22.    End
23. % Signaler l'état actuel des ressources avec la variable Flag
24. c=1; k=1;
25. while c<length(data1(:,1))
26.    if data1(c+1,1)<data1(c,1)
27.       flag(c,1)=-1;
28.    elseif data1(c+1,1)>data1(c,1)
29.       flag(c,1)=1;
30.    elseif data1(c+1,1)==data1(c,1)
31.       flag(c,1)=0;
32.    End
33.    c=c+1;
34. End
35. while k<length(data1(:,2))
36.    if data1(k+1,2)<data1(k,2)
37.       flag(k,2)=-1;
38.    elseif data1(k+1,2)>data1(k,2)
39.       flag(k,2)=1;
40.    elseif data1(k+1,2)==data1(k,2)
41.       flag(k,2)=0;
42.    end
43.    k=k+1;
44. End
45.  %Signaler le besoin d'un ajustement
```

Algorithme 4.4 Adaptation de ressources en utilisant le Mahalanobis (Suite)

```
46. h=1;
47. while h<=length(out_MD)
48.     if flag(pos_OMD(h),1)==1 && flag(pos_OMD(h),2)==1
49.           signal(h,:)=[1,pos_OMD(h)];
50.     elseif (flag(pos_OMD(h),1)==1 && flag(pos_OMD(h),2)==-
      1)||(flag(pos_OMD(h),1)==-1 && flag(pos_OMD(h),2)==1)
51.           signal(h,:)=[0,pos_OMD(h)];
52.     elseif flag(pos_OMD(h),1)==-1 && flag(pos_OMD(h),2)==-1
53.           signal(h,:)=[-1,pos_OMD(h)];
54.     End
55.     h=h+1;
56. end
57.  % Faire la décision d'ajustement des ressources
58. k=1;
59. while k<length(out_MD)
60.       if (valR(k,1)>=30 || R(k)==1) && signal(k,1)==1
61.            decision{k}='ajout';
62.          confIn(:,1) = confIn(:,1)+1 %Actualiser le nombre des cœurs disponibles
63.          confIn(:,2) = confIn(:,2)*2 %doubler la capacité de la 2ème métrique
64.       elseif (valR(k,1)>=30 ||  R(k)==1) && signal(k,1)==-1
65.          decision{k}='suppression';
66.          confIn(:,1) = confIn(:,1)-1 %Actualiser le nombre des cœurs disponibles
67.     End
68. %Afficher la figure de la position des outliers par rapport à leurs valeurs
      réelles
69. Return decision, confIn
70. End
```

4.3 Analyse des performances de l'algorithme d'adaptation des ressources

Dans cette section, nous allons analyser les performances de l'algorithme proposé en l'appliquant sur les différents scénarios décrits dans la section 2.3. Nous présentons dans ce qui suit les différents scénarios où nous avons effectué une adaptation de ressources. Les autres scénarios, où il n'y a pas eu un ajustement de ressources, seront représentés dans l'annexe IV. À cause des données disponibles, nous avons fixé un seuil de 30% pour la consommation de CPU et de 50% pour la bande passante.

4.3.1 Scénario 1

Dans ce scénario, nous avons considéré les données IMS décrites dans la table 2.2. Avec les deux méthodes de détection des outliers, aucune valeur aberrante n'est identifiée et ceci pour les deux métriques. Par conséquent, aucune adaptation de ressource n'est nécessaire pour ce scénario de déploiement. La figure 4.1 montre l'allure de la distance Mahalanobis et la variation des données : Nous remarquons que la distance MD est faible pour les premières données observées et augmente à la position 70 jusqu'à 73. Cependant, la valeur maximale de CPU est de 50% pour un seul cœur et la variation de *throughput* n'est pas importante. Ces valeurs ne sont pas critiques. Ainsi, le système n'a pas besoin de plus de ressource et nous n'avons pas remarqué de sous-utilisation en termes de CPU.

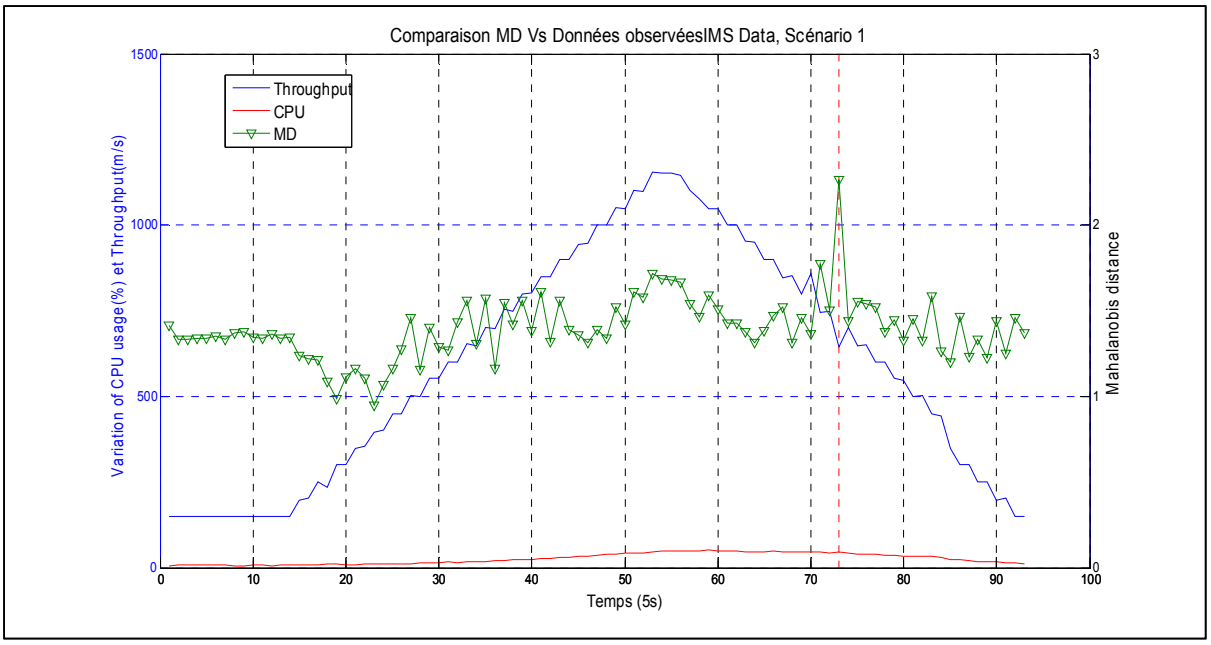

Figure 4.1 Scénario 1- Résultat de l'algorithme d'adaptation avec le Mahalanobis

4.3.2 Scénario 3

Dans ce scénario, la valeur maximale de CPU des outliers identifiées est égale à 37.66% relevée à la position 101 correspondant à l'instant 505 secondes. Cette valeur n'est pas critique, mais sa variation est importante.

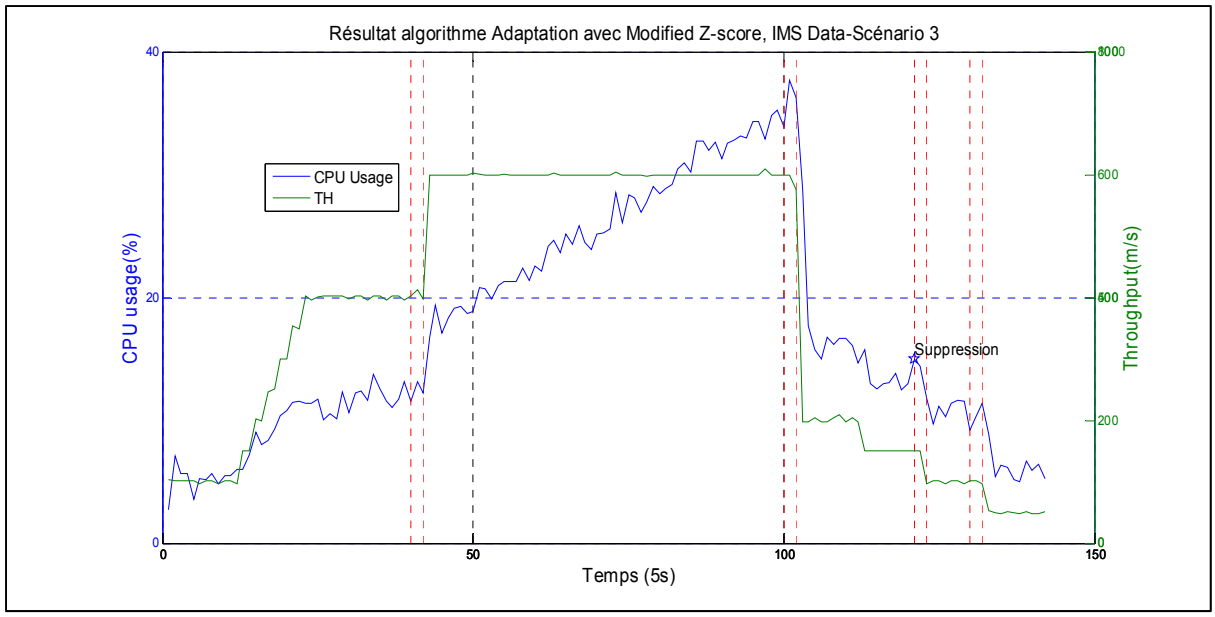

Figure 4.2 Scénario 3-Résultat de l'algorithme d'adaptation avec le Modified Z-score

On remarque 4 paires d'outliers avec le Modified Z-score pour le deux métriques étudiées (le CPU et le TH) et représentés par les lignes rouges pointillées dans la figure 4.3. L'analyse de ces outliers est décrite dans le chapitre 2. Étant donné que la valeur de CPU n'a pas dépassé le 40% d'une part et d'autre part on se retrouve avec une chute des valeurs après cette augmentation, notre algorithme n'a pas enregistré un besoin d'ajustement de ressource à ce moment. Cependant, le modified Z-score a signalé une suppression de ressources à la position 123. Cette décision est prise en retard puisque la grande chute de ressources s'est passée à la position 103.

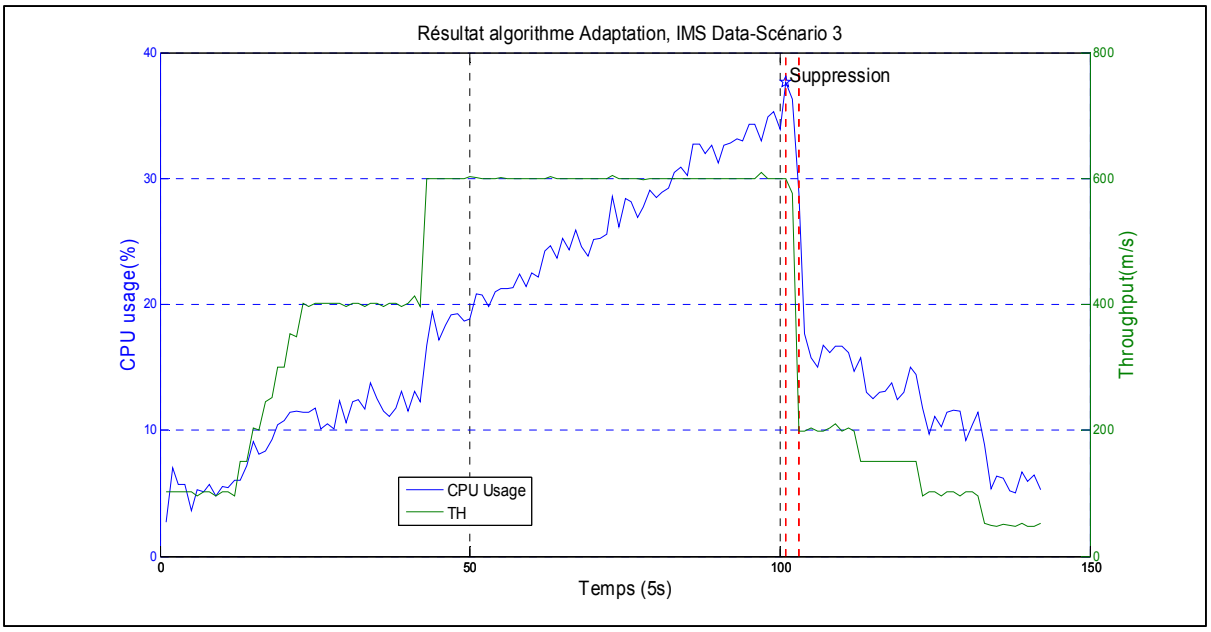

Figure 4.3 Scénario 3-Résultat de l'algorithme d'adaptation avec le Mahalanobis

Le Mahalanobis a enregistré deux outliers, un au début de la chute de données et l'autre après la chute. Ces deux outliers sont jugés comme des vrais outliers puisqu'ils nous indiquent le moment exact où un ajustement doit être effectué. Cependant avec la valeur de CPU, 36.6%, l'algorithme d'adaptation peut nous indiquer une insuffisance de ressources à l'instant t suite au seuil fixé à 30% comme illustré dans la figure 4.3. Cette adaptation n'a pas été effectuée puisque le système subit une diminution de la consommation de ressources à l'instant t+1.

4.3.3 Scénario 6

Pour ce scénario, l'algorithme identifie 2 suppressions pour la métrique CPU illustrée dans la figure 4.4 mais aucune adaptation n'a été signalée pour la bande passante. Nous remarquons que la première suppression notifiée à l'instant t par la méthode de modified Z-score n'est pas correcte. Cette décision vient de fait que l'outlier considère la diminution légère de la valeur a la position t+1. Cependant, le système aura besoin de plus de ressources pour confronter les fluctuations de ressources. Ainsi la deuxième suppression peut être considérée comme une bonne décision parce que nous avons une chute dans les valeurs observées. Nous pouvons conclure le modified Z-score n'est pas adapté au système qui subisse beaucoup de variations.

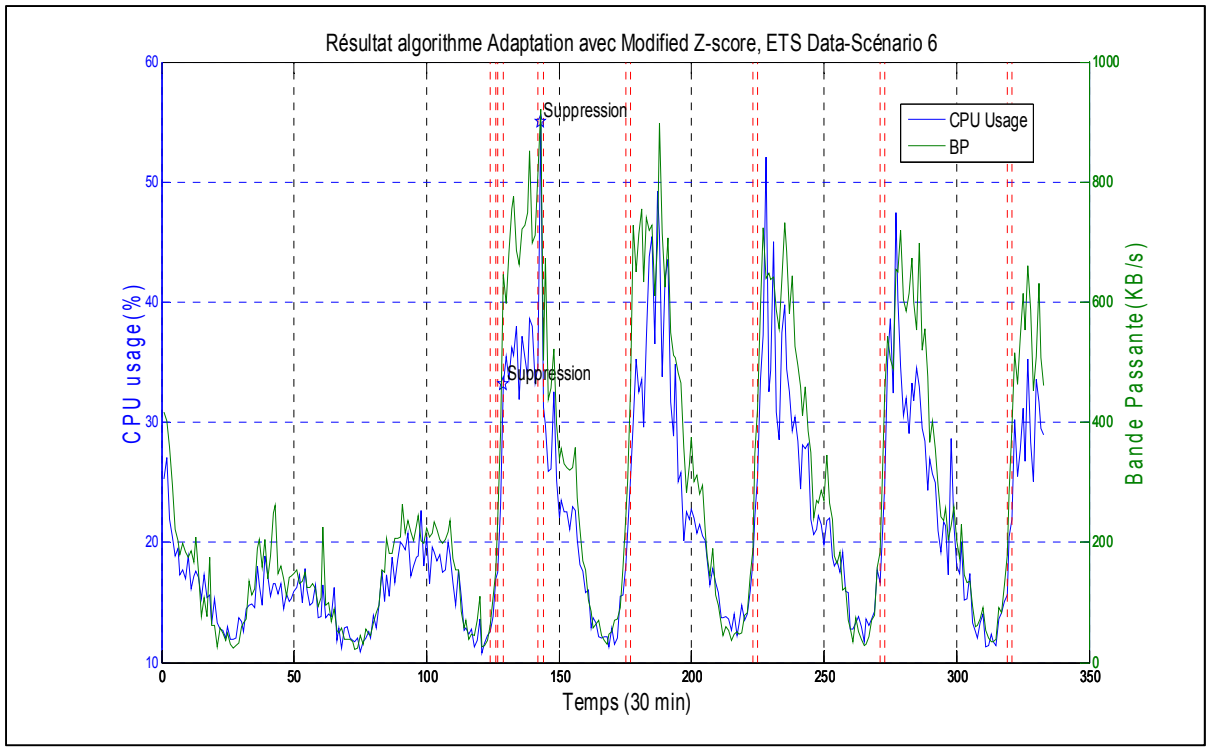

Figure 4.4 Scénario 6-Résultat de l'algorithme d'adaptation avec le Modified Z-score

La figure 4.5 représente les résultats obtenus en appliquant le Mahalanobis dans notre algorithme d'adaptation. Nous remarquons que pour chaque cycle nous avons un ajout et une suppression de ressources. Seul le CPU sera ajusté, la bande passante n'a pas dépassé les 50%. Chaque ajout illustré dans la figure 4.5 est signalé au début de chaque cycle. Ainsi la suppression a été effectuée au moment où les données diminuent. Seule le troisième cycle (représenté par le rectangle rouge) n'a pas subi une adaptation à cause des valeurs réelles collectées.

Avec l'intervalle de temps de 30 minutes, un ajout de ressources est jugé important pour que le système assure le bon fonctionnement de ses tâches. Le mécanisme d'adaptation de ressources propose d'ajouter à chaque fois un seul cœur pour le CPU. Ainsi, la configuration de système change pour les prochaines décisions. La suppression d'un cœur à la fin du premier cycle est jugée nécessaire puisque le système subit une diminution dans la consommation de ressources de la position 142 jusqu'à la position 175 pour le 1er cycle. Les mêmes explications sont valables pour le 2ème et le 4ème cycle.

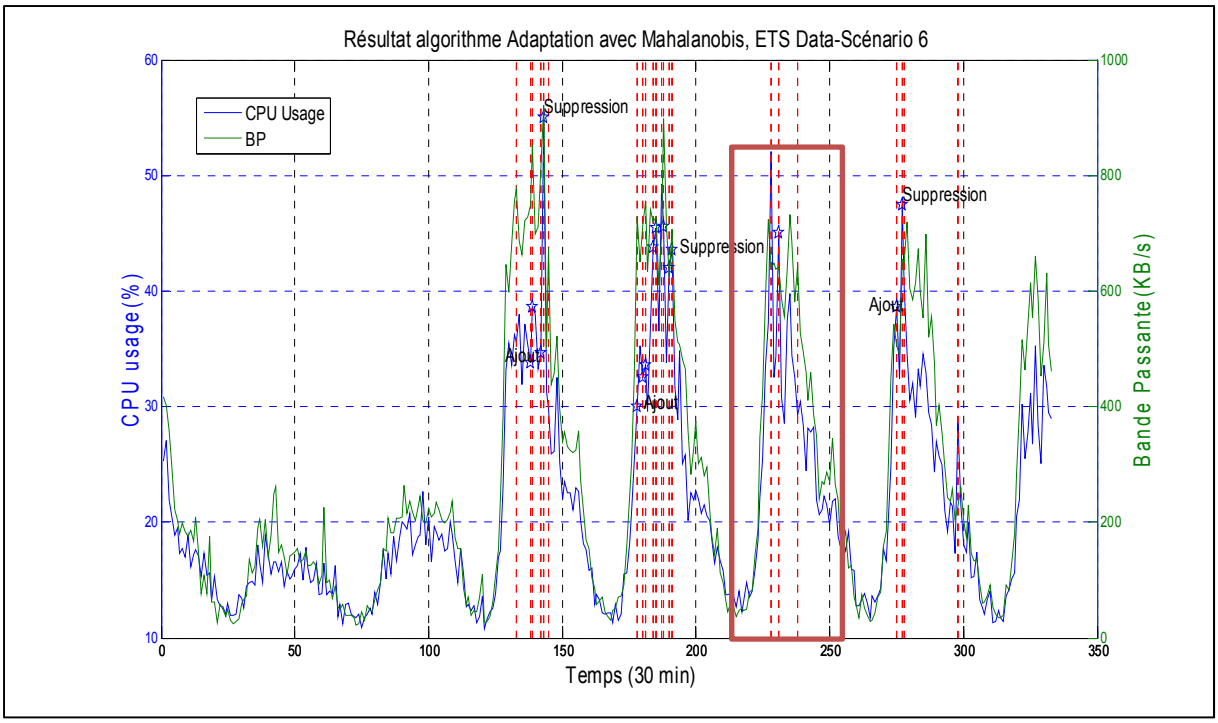

Figure 4.5 Scénario 6-Résultat de l'algorithme d'adaptation avec le Mahalanobis

4.3.4 Scénario 9

Pour le modified Z-score, il n'y a pas de correspondance entre les outliers c'est pourquoi l'algorithme n'a pas pu identifier un besoin d'ajustement de ressources.

La figure 4.6 montre le résultat obtenu en appliquant le Mahalanobis. Nous remarquons un évènement inhabituel. La consommation de CPU à la position 3 est de 400%. Selon la fiche technique de données de google, les valeurs collectées et publiées représentent la consommation d'un seul cœur pour une machine virtuelle. Cependant, une grande partie de la configuration du système reste cachée pour les chercheurs et les utilisateurs. Plusieurs explications peuvent justifier cette valeur tels que la migration des machines virtuelles ou l'existence d'un mécanisme d'adaptation de ressources qui permettent à une machine d'avoir au besoin recourt à d'autres ressources.

Nous nous intéressons au résultat obtenu par notre algorithme d'adaptation de ressources avec le Mahalanobis. Nous remarquons 3 ajouts proposés par l'algorithme d'adaptation. Nous

ajoutons un cœur au moment du premier signal d'ajustement. La consommation de CPU est fluide et ne subit pas des grandes variations ou des chutes de consommation. L'algorithme d'adaptation propose 2 ajouts à la position 30 et 75. À ce moment, il vérifie la capacité actuelle du système et l'état des ressources disponibles, si les ressources sont suffisantes (Dans notre cas, nous avons ajouté 1 cœur donc le système possède présentement de 2 cœurs en termes de CPU) aucun ajustement ne sera effectué.

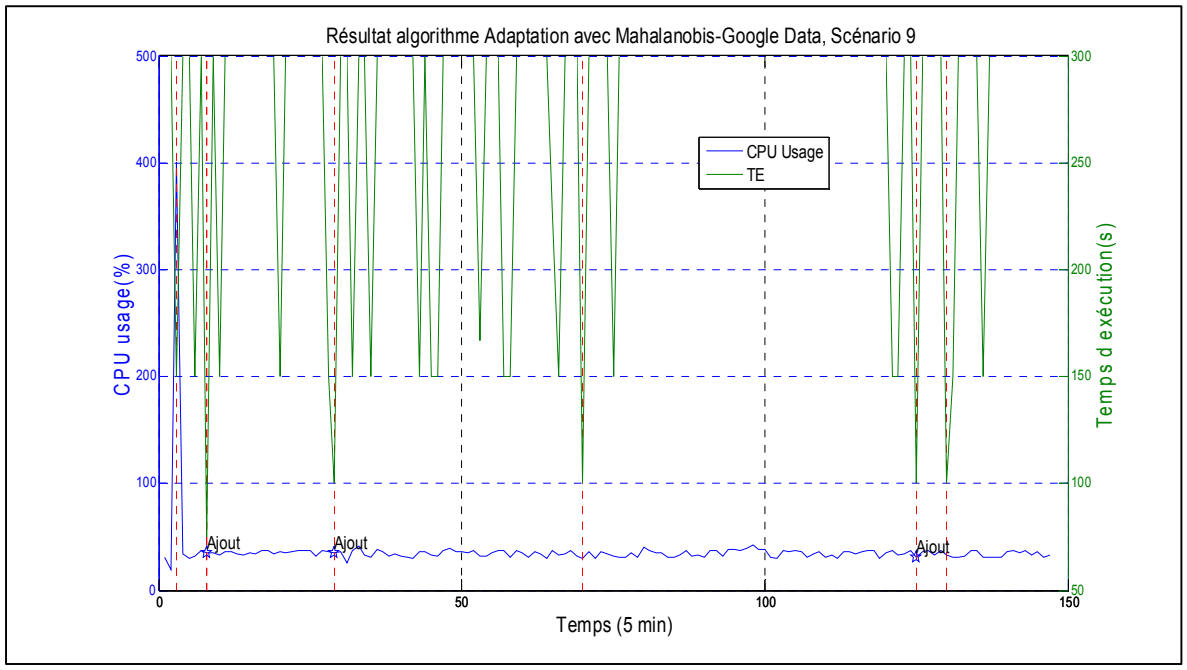

Figure 4.6 Scénario 9-Résultat de l'algorithme d'adaptation avec le Mahalanobis

4.3.5 Scénario 11

Dans ce scénario, nous remarquons que l'algorithme nous indique une possibilité d'adapter les ressources avec la méthode de modified Z-score. La figure 4.7 nous indique un ajout et trois suppressions.

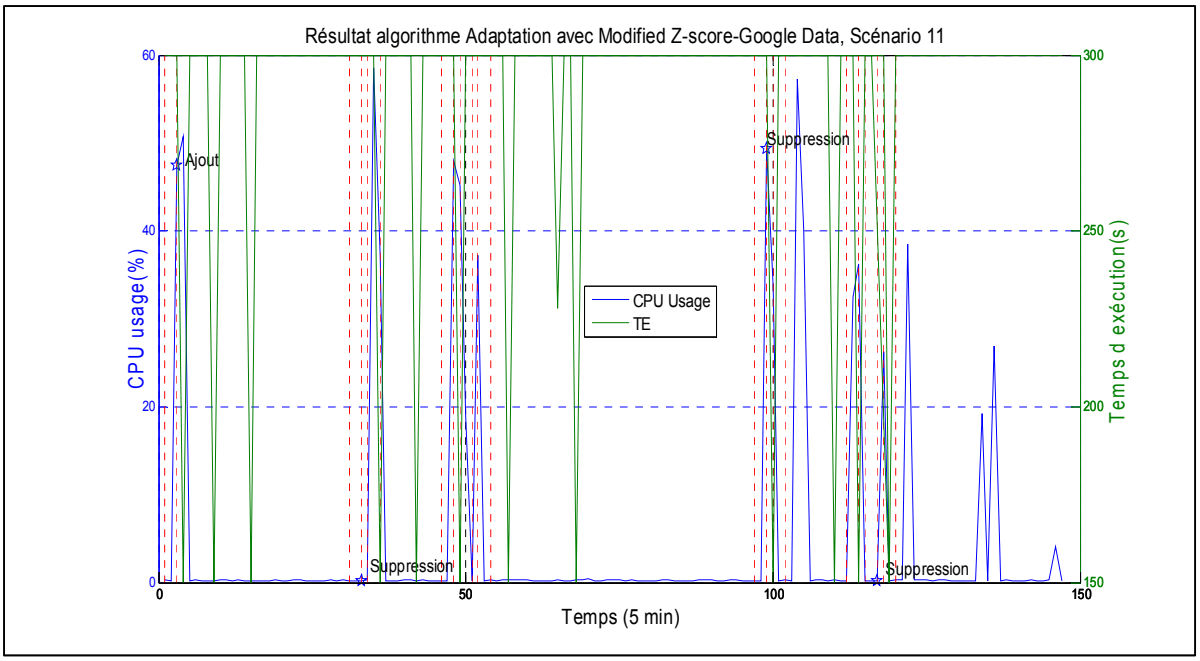

Figure 4.7 Scénario 11-Résultat de l'algorithme d'adaptation avec le modified Z-score

D'après les variations des données observées, nous jugeons que les résultats obtenus en utilisant le modified Z-score sont incorrectes. Le premier ajout a été détecté après l'augmentation des valeurs, en plus, les données diminuent juste après cette augmentation.

La première suppression et la troisième sont identifiées avant une augmentation de la consommation de CPU, ce résultat n'est pas correct et n'est pas logique. La deuxième suppression peut être considérée comme une bonne décision dans le cas où le système subit une diminution de ressources pour une période de temps, sauf que dans la figure 4.7 nous remarquons qu'il y aura une augmentation des ressources à l'instant t+2.

Quant aux résultats obtenus par le Mahalanobis, nous remarquons qu'aucune adaptation n'a été proposée. Ceci peut être expliqué par les fluctuations successives de la consommation de ressources pour ce scénario illustré dans la figure 4.8.

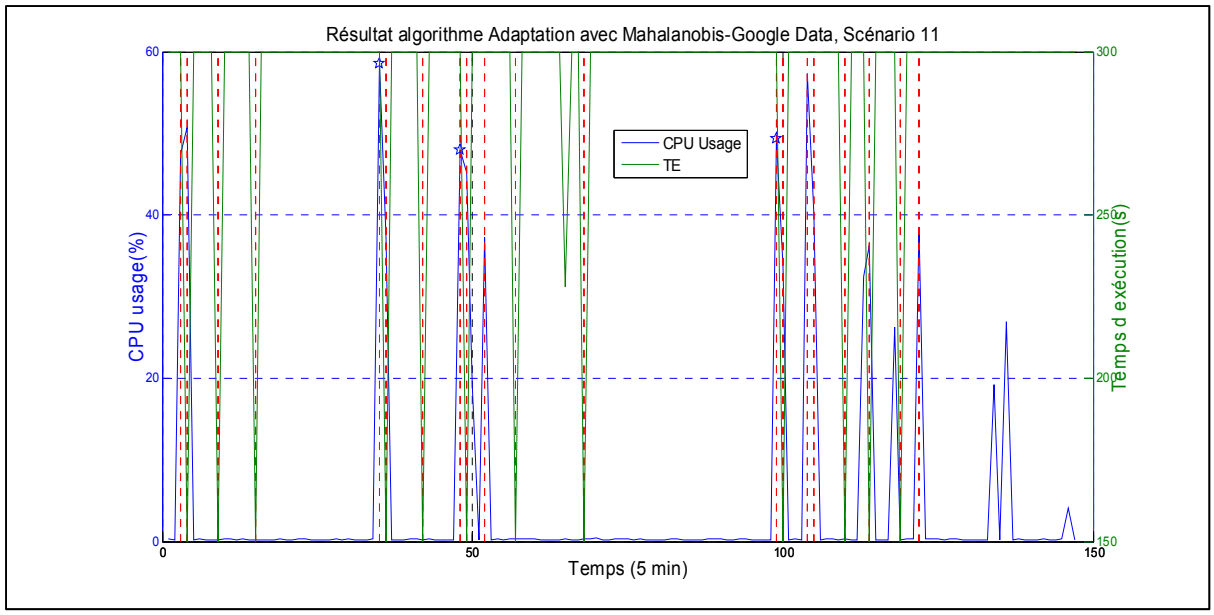

Figure 4.8 Scénario 11-Résultat de l'algorithme d'adaptation avec le Mahalanobis

4.4 Récapitulatif des performances de l'algorithme d'adaptation des ressources

Dans cette partie, nous exposons dans le tableau 4.1 les différents résultats obtenus de l'adaptation de ressources en utilisant les deux méthodes de détection d'outlier à savoir, le *modified Z-score* et le *Mahalanobis*.

Tableau 4.1 Récapitulatif des performances de l'algorithme d'adaptation des ressources

Scénarios	Modified-Zscore	Mahalanobis	Remarques
Scénario1	Aucune adaptation.		Aucun outlier n'a été détecté avec les 2 méthodes utilisées. (Pas de correspondance entre les deux métriques)
Scénario2	Aucune adaptation.		Le besoin d'augmentation de ressources signalé est suivi d'une chute dans les valeurs observées de la métrique c'est pour cette raison qu'aucune adaptation n'est nécessaire pour ce scénario.
Scénario3	Décision prise au mauvais moment.	Décision signalée au bon moment.	Les deux méthodes utilisées ont signalé une suppression de ressources lors de la chute de ressources. Le Mahalanobis a

Tableau 4.1 Récapitulatif des performances de l'algorithme d'adaptation des ressources (Suite)

			détecté le moment exact où le système aura besoin de moins de ressources.
Scénario4	Les deux méthodes proposent une suppression en termes de CPU.		La suppression proposée n'a pas été effectuée à cause de ressources disponibles (la configuration initiale de ce jeu de données est un seul cœur pour le CPU).
Scénario5	La consommation de CPU n'a pas dépassé le 30%.		Un seuil de 30% pour la consommation de CPU a été fixé pour signaler un besoin d'ajustement. Ce seuil n'a pas été dépassé dans ce scénario.
Scénario6	2 suppressions signalées. Une au bon moment et une au mauvais moment.	3 ajouts et 3 suppressions sont signalés.	Ce scénario est cyclique. Le Modified Z-score propose 2 suppressions, une au moment où les données diminuent et une au moment d'une augmentation. Le Mahalanobis propose un ajout au début de cycle où la consommation de CPU et de la BP augmente et une suppression à la fin de chaque cycle lorsque la valeur des données diminue.
Scénario7	Le CPU pour les outliers détectés n'a pas dépassé le 30%.		Même remarque que le scénario 5.
Scénario8			
Scénario9	Pas de correspondance entre les *outliers* de deux métriques	3 ajouts signalés par le Mahalanobis.	Le Mahalanobis nous a permis de détecter les moments où le système a besoin d'ajustement de ressources contrairement au Modified Z-score où nous n'avons pas pu détecter des outliers. L'algorithme d'adaptation nous a permis de contrôler et décider les moments et la quantité de ressources à ajuster. 1 cœur a été ajouté dans ce scénario pour la métrique CPU.
Scénario10	Aucune adaptation.		Même remarque que le scénario 2.
Scénario 11	1 ajout et 3 suppressions	Aucune adaptation proposée	La décision d'adaptation est affectée par les fluctuations des données.

Le tableau 4.1 résume les différentes observations tirées des tests effectués sur les environnements virtualisés en utilisant notre algorithme d'adaptation.

Conclusion

Dans ce chapitre, nous avons présenté une approche permettant une adaptation des ressources pour les systèmes virtualisés. Cette solution est basée sur la méthode de détection des outliers, appelée distance de Mahalanobis.

Le Mahalanobis est une méthode robuste pour les données représentant des grandes variations. Elle est basée sur la corrélation de données et permet d'identifier les valeurs aberrantes. Nous l'avons comparé avec le Modified Z-score et elle nous donne des meilleurs résultats.

CONCLUSION

La gestion des ressources dans des environnements virtualisés est devenue un vrai défi pour les fournisseurs d'infrastructures cloud. La nature élastique du cloud permet de faciliter le partage des ressources, d'offrir un service diversifié, une meilleure exploitation des ressources et une réduction des coûts. Cependant, avec tous ces avantages, les systèmes virtualisés rencontrent de nombreux défis.

L'objectif principal des fournisseurs d'infrastructure cloud est d'assurer une bonne qualité de service à leurs clients. Les infrastructures de communication sont généralement sur-approvisionnées, conduisant ainsi à une sous-utilisation des ressources en dehors des heures de pointe et surtout des coûts de gestion, d'opération et de consommation d'énergie élevés. Il s'agit donc de trouver une stratégie efficace qui permet d'ajuster l'approvisionnement en ressources selon la demande tout en garantissant la qualité de service dans des environnements virtualisés.

Dans le cadre de cette recherche, nous avons focalisé notre travail sur l'un des défis importants auquel font face les fournisseurs d'infrastructures cloud à savoir la gestion efficace des ressources. Ainsi dans le cadre de ce projet, nous avons défini et testé un mécanisme fiable d'adaptation des ressources dans des environnements virtualisés sensible à la qualité de service. Nous avons utilisé comme plateforme de tests plusieurs types de systèmes virtualisés à savoir IMS, les grappes de google et le cloud de l'ÉTS.

Notre méthodologie de travail consiste d'abord à collecter des données de différents systèmes. Cette étape nous a permis d'analyser le comportement des systèmes étudiés selon les scénarios de déploiement, les profils de charges, leurs configurations et leurs conceptions. Ensuite, nous avons défini et testé des algorithmes de détection des valeurs aberrantes permettant d'identifier les moments où le système a besoin de réajustement de ressources. Finalement, nous avons développé un algorithme permettant d'adapter les ressources selon les besoins du système

analysé en se basant sur la méthode de détection d'outliers appelée « la distance Mahalanobis ».

Comme travaux futurs, nous proposons de tester le mécanisme d'adaptation de ressource proposé dans des environnements virtualisés réels et de l'adapter à différentes contraintes imposées par un tel environnement telles que l'hétérogénéité des ressources et des exigences de QoS. Pour cela, nous pensons qu'il serait intéressant d'implémenter cette solution dans un environnement virtualisé hétérogène. Ainsi, nous proposons d'implémenter une solution permettant la collecte et la surveillance des métriques en temps réels comme module complémentaire à notre solution et de considérer la corrélation entre plusieurs métriques (3 métriques et plus).

ANNEXE I

TÉLÉCHARGEMENT DES DONNÉES DE GOOGLE

Pour télécharger les données de google, il faut tout d'abord préparer l'environnement :

- Télécharger miniconda3
- Installer python version 2.7
- Créer la variable d'environnement*:* ***conda create -n*** *conda_en* ***anaconda python***
- Activer la variable d'environnement : **activate** *conda_en*
- Installer pandas *:* ***conda install pandas***
- Tester pandas :

```
>>> import pandas as pd
>>> pd.test()
```

- Exécuter le script de téléchargement (extractData.py)
- Une fois les données téléchargées, il faut filtrer et classifier les données à analyser.

La table utilisée dans les données de google est **Task usage**. C'est la table de données ayant la plus d'information sur les taches, les jobs et les machines virtuelles des applications de google 2011.

Algorithme I.1 Extraction des données de google

extractData.py

```
import scipy.io as sio
from random import randint, sample, seed
import numpy as np
from os import listdir, chdir
chdir('C:\GoogleData\clusterdata-2011-2')
print((listdir('.')))
print((sorted(listdir('task_events'))[-1]))
import pandas as pd
from pandas import read_csv
from os import path
```

Algorithme I.1 Extraction des données de google (Suite)

```
task_event_df = read_csv(path.join('task_events', 'part-00498-of-00500.csv.gz'), header=None, index_col=False,
compression='gzip', names=task_events_csv_colnames)
sample_moments = sorted(sample(list(range(133405)), 200))
print(sample_moments)
mydata=[[0 for j in range(2)] for i in range(200)]
g=0
for indice in sample_moments:
mydata[g][0] = task_event_df['time'][indice]
mydata[g][1] = task_event_df['cpu_requested'][indice]
g+=1
times = [row[0] for row in mydata]
times_stamp = [int(x) for x in times]
cpus = [row[1] for row in mydata]
print (np.array(times_stamp).T)
print (np.array(cpus).T)
sio.savemat('test498.mat',
dict(timestamp=np.array(times_stamp).T, CPU_req=np.array(cpus).T ))
print (liste1)
print (tpsCPUdicti)
arrayCPU = np.asarray(tpsCPUdicti)
print(arrayCPU)
```

Algorithme I.2 Filtrer les données de google

FilterData.py

```
load('C:\Users\an41560\Desktop\GoogleData-MV\clusterdata-2011-2\test_495_usage_ext.mat')
dataset = [double(job_495) double(task_495) cpu_495 double(Stime_495) double(Etime_495) double(machine_495)];

i=1; j=1; s=1; nbM=[];task_nb=1;
job=dataset(:,1);
task=dataset(:,2);
cpu=dataset(:,3);
stime=dataset(:,4);
etime=dataset(:,5);
machine=dataset(:,6);
%extract job deployed on one machine
while (i<= length(job))
   time=etime(i)-stime(i);
```

Algorithme I.2 Filtrer les données de google (Suite)

```
    j= i+1; nbm=1; nbj=1;

   while (j<=length(job))
     if (job(i) == job(j) && i~=j)
       nbj=nbj+1;
       if (machine(i) == machine(j) && task(i) == task(j))
         nbm=nbm+1;
         task_nb=task_nb+1;
         time=time+(etime(j)-stime(j));
       end
     end
       j=j+1;
   end
   M1{s}=job(i);
   M2{s}=machine(i);
   M3{s}=task(i);
   M4{s}=nbm;
   M5{s} =task_nb;
   M6{s} =time*1e-6;
   task_nb=1;
   nbm=1;
    i=i+1;
    s=s+1;
end

nbM = horzcat(M1',M2',M3',M4',M5',M6');

i=1; j=1;
while (i<= length(nbM(:,1)))
    j= i+1;
   while (j<=length(nbM(:,1)))
    if (cell2mat(nbM(i,1)) == cell2mat(nbM(j,1)) && i~=j)
         nbM(j,:)=[];
         j=j-1;
    end
        j=j+1;
   end
    i=i+1;
end
```

ANNEXE II

L'IMPACT DE LA PROPABILITÉ SUR LA DISTANCE MAHALANOBIS

La valeur de la probabilité confirmée par les travaux antérieurs est de 95% pour 2 variables. En changeant la valeur de probabilité de notre ellipse de confiance, nous remarquons qu'en diminuant la valeur nous avons plus des valeurs en dehors de l'ellipse et qu'en augmentant nous avons moins. Cependant, même avec 99%, nous avons au moins une valeur qui sort de l'ellipse dans presque tous les scénarios IMS et cette valeur enregistre la plus grande valeur de MD.

Nous exposons quelques figures afin de montrer l'effet de l'intervalle de confiance sur les outliers détectés avec le MD sur les différents scénarios étudiés de google.

- **IMS scénario 1 :**

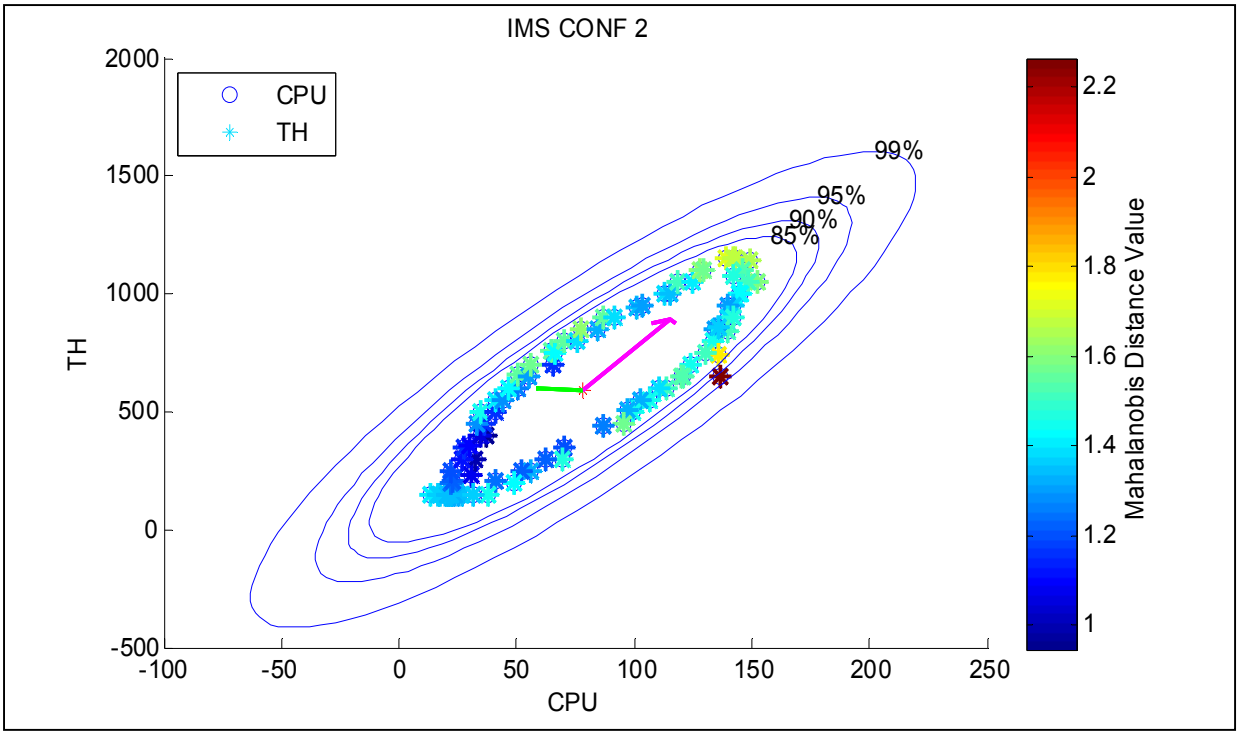

Figure II.1 : Résultat de l'application de Mahalanobis en changeant la probabilité

- **IMS scénario 2 :**

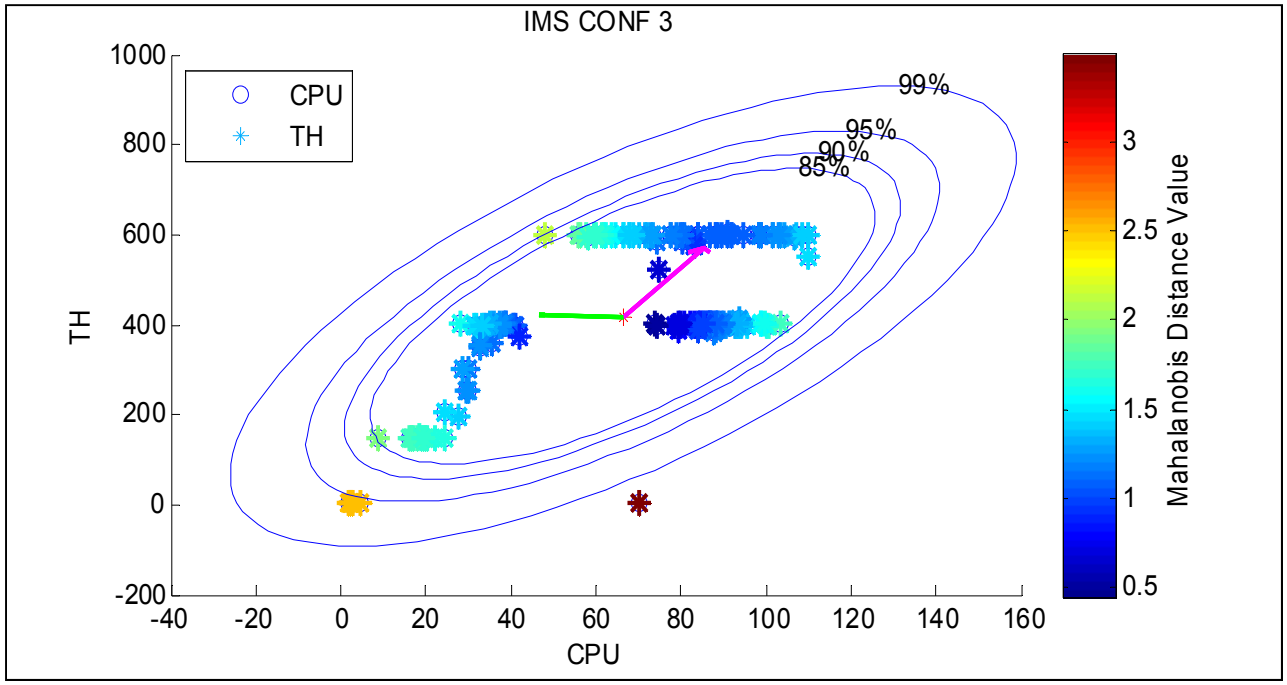

Figure II.2 : Résultat de l'application de Mahalanobis en changeant la probabilité

- **IMS scénario 3 :**

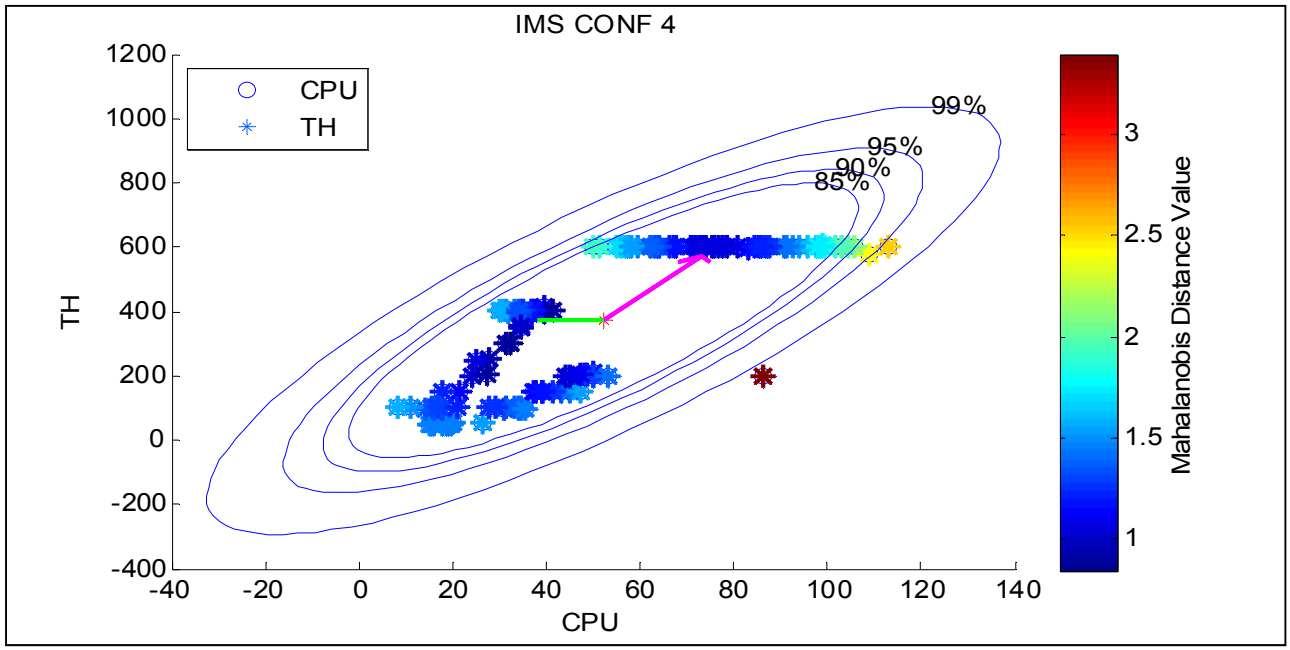

Figure II.3 : Résultat de l'application de Mahalanobis en changeant la probabilité

- **IMS scénario 4 :**

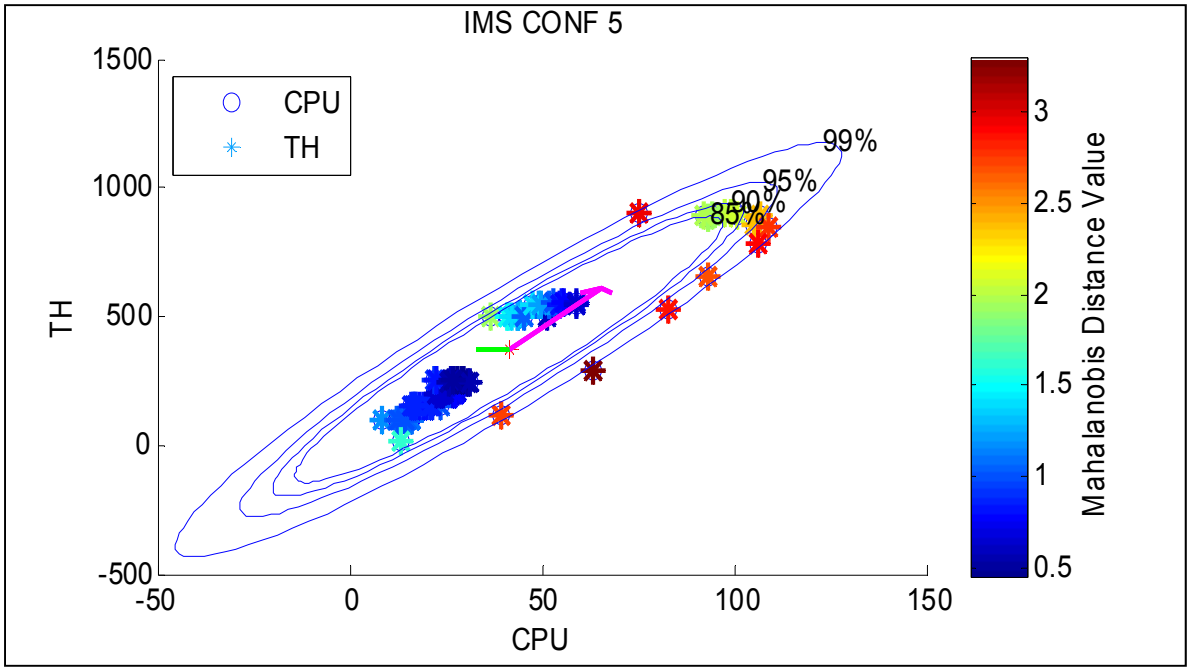

Figure II.4 : Résultat de l'application de Mahalanobis en changeant la probabilité

ANNEXE III

L'INFLUENCE DU SEUIL ET DE LA TAILLE DE DONNÉES SUR LES RÉSULTATS OBTENUS AVEC LE MODIFIED Z-SCORE

Scénarios concrets d'IMS

❖ Scénario de test 1

Dans ce scénario de test, nous avons collecté 14 sous-ensembles avec un intervalle de 12 valeurs.

- **Seuil = 3.5**

Le tableau III.1 représente les résultats obtenus avec un intervalle de 12 valeurs et un seuil de 3.5.

Tableau III.1 : Tableau représentant les outliers détectés pour IMS avec un seuil de 3.5

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
4	3.5	1/	14	4.6466	[-1.8486,4.6466]	0.1017
9	3.5	2/	14	4.8009	[-34.3201,4.8009]	-1.0293
			34	-34.3201		
10	3.5	5/	14	4.0138	[-28.6939,11.2675]	-0.4307
			34	-28.6939		
			35	-8.5695		
			37	11.2675		
			38	11.1348		

- **Seuil = 3**

Le tableau III.2 représente les résultats obtenus avec un intervalle de 12 valeurs et un seuil de 3.

Tableau III.2 : Tableau représentant les outliers détectés pour IMS avec un seuil de 3

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
4	3	1/	14	4.6466	[-1.8486,4.6466]	0.1017
9	3	2/	14	4.8009	[-34.3201,4.8009]	-1.0293
			34	-34.3201		
10	3	5/	14	4.0138	[-29.2508, 11.3766]	-0.4843
			34	-28.6939		
			35	-8.5695		
			37	11.2675		
			38	11.1348		
12	3	6/	14	4.2413	[-29.5751,11.7409]	-0.3892
			34	-29,5751		
			35	-8,7685		
			37	11.7409		
			38	11.6037		
			46	-3.1896		
13	3	5/	14	3.7098	[-25.6310, 10.2167]	-0.3218
			34	-25.6310		
			35	-7.5782		
			37	10.2167		
			38	10.0977		

- **Seuil = 4**

Le tableau III.3 représente les résultats obtenus avec un intervalle de 12 valeurs et un seuil de 4.

Tableau III.3 : Tableau représentant les outliers détectés pour IMS avec un seuil de 4

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
4	4	1/	14	4.6466	[-1.8486, 4.6466]	0.1017
9	4	2/	14	4.8009	[-34.3201, 4.8009]	-1.0930
			34	-34.3201		
10	4	5/	14	4.01382	[-28.6939, 11.2675]	-0.4307
			34	-28.6939		
			35	-8.5695		
			37	11.2675		
			38	11.1348		

Tableau III.3 : Tableau représentant les outliers détectés pour IMS avec un seuil de 4 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
13	4	4/	34	-25.6310	[-25.6310, 10.2167]	-0.3218
			35	-7.5782		
			37	10.2167		
			38	10.0977		

❖ **Scénario de test 2**

Dans ce scénario de test, nous augmentons la taille de l'échantillon. Dans le scénario de test précédant nous avons choisi 12 valeurs et pour ce scénario nous allons prendre un échantillon de 18 valeurs. Le changement de l'intervalle de test nous aide à mieux voir l'impact de l'ensemble de données sur la valeur de modified z-score.

Nous avons collecté 9 sous-ensembles avec un intervalle de 18 valeurs. Le choix de la taille de l'échantillon vient de faite que la variation calculée par le modified z-score est basé sur 3 valeurs. Donc la taille de l'échatilon à prendre doit etre un multiple de 3.

- **Seuil = 3.5**

Aucun outlier n'a été détecté jusqu'à le sous-ensemble numéro 3 (c'est-à-dire après 54 valeurs observés). Les scores obtenus pour ces sous-ensembles sont inférieurs au seuil choisi. Le premier outlier détecté a été obtenu au sous-ensemble 3. Le tableau III.4 montre les différents scores des outliers détectés avec leurs positions dans le temps. L'unité de temps de modified zscore pour IMS est de 15s.

Pour le reste des sous-ensembles 8 et 9, il n'y a pas de nouveau outlier, ils gardent les mêmes outliers detectés dans le sous-ensemble 7. Cependant la valeur de mzscore pour ces outliers change. On remarque une augmentation de la valeur puis une diminution. Cette variation de score dépend de la valeur des données observés et leurs variations dans le temps.

Tableau III.4 : Tableau représentant les outliers détectés pour IMS avec un seuil de 3.5

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
3	3.5	1/	14	4.9161	[-1.8289,4.9161]	0.1508
6	3.5	2/	14	4.8009	[-34.3201, 4.8009]	-1.0293
			34	-34.3201		
7	3.5	5/	14	4.0584	[-29.7580, 11.5580]	-0.4819
			34	-29.7580		
			35	-8.9514		
			37	11.5580		
			38	11.4208		

- **Seuil = 3**

Nous remarquons que les scores des outliers détectés pour un échantillon de 18 valeurs dépasse largement le 3,5, cependant, aucun autre outlier n'a été detecté pour un seuil de 3.

Dans le tableau III.5, nous avons remarqué l'apparition d'un nouveau outlier et 2 nouveaux sous-ensembles, par contre, en augmentant la taille de l'échantillon de 12 à 18 pour le même seuil, nous n'avons pas obtenu les mêmes résultats. On peut déduire alors que la valeur de modified z-score est impacté par la taille des données traitées.

Tableau III.5 : Tableau représentant les outliers détectés pour IMS avec un seuil de 3

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
3	3	1/	14	4.9161	[-1.8289,4.9161]	0.1508
6	3	2/	14	4.8009	[-34.3201, 4.8009]	-1.0293
			34	-34.3201		
7	3	5/	14	4.0584	[-29.7580, 11.5580]	-0.4819
			34	-29.7580		
			35	-8.9514		
			37	11.5580		
			38	11.4208		

- **Seuil = 4**

En augmentant le seuil, on remarque dans le tableau III.6 la disparition de premier outlier détecté qui est à la position 14 dans le dernier sous-ensemble.la valeur de cet outlier dans le sous-ensemble 9 est de 3.603 < 4. Nous remarquons aussi que le score des outliers détectés diminue en augmentant la taille de données traitées et cette remarque est aussi valable pour les restes des outliers. Donc la variation de données en augmentant la taille diminue de plus en plus.

Tableau III.6 : Tableau représentant les outliers détectés pour IMS avec un seuil de 4

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
3	4	1/	14	4.9161	[-1.8289,4.9161]	0.1508
6	4	2/	14	4.8009	[-34.3201, 4.8009]	-1.0293
			34	-34.3201		
7	4	5/	14	4.0584	[-29.7580, 11.5580]	-0.4819
			34	-29.7580		
			35	-8.9514		
			37	11.5580		
			38	11.4208		
9	4	4/	34	-24.8986	[-24.8986, 9.9248]	-0.3020
			35	-7.3617		
			37	9.9248		
			38	9.8092		

❖ **Scénario de test 3**

Dans ce scénario, nous allons diminuer la valeur de notre échantillon et nous allons comparer les résultats obtenus avec les résultats de scénario de test 1 et 2. Nous avons collecté 28 sous-ensembles avec un intervalle de 6 données.

- **Seuil = 3.5**

Nous remarquons dans cette partie l'apparition d'un nouveau sous-ensemble avec 3 outliers. Nous avons obtenu dans ce scénario plus de sous-ensemble que pour les autres scénarios représentés dans la table III.7. En ayant des petits sous-ensembles, on peut déduire l'ensemble

des données qui nous permet d'avoir ces outliers. En ayant un grand nombre de données traités comme dans le scénario 1 et 2, nous n'avons jamais eu un sous-ensemble avec 3 outliers.

Tableau III.7 : Tableau représentant les outliers détectés pour IMS avec un seuil de 3.5

Numéro sous-ensemble	Seuil	Nombre Outliers/position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
8	3.5	1/	14	4.6466	[-1.8486,4.6466]	0.1017
18	3.5	2/	14	4.8009	[-34.3201, 4.8009]	-1.0293
			34	-34.3201		
19	3.5	3/	14	4.4686	[-31.1595,4.4686]	-1.0767
			34	-31.1595		
			35	-9.2382		
20	3.5	5/	14	4.0138	[-28.6939, 11.2675]	-0.4307
			34	-28.6939		
			35	-8.5695		
			37	11.2675		
			38	11.1348		

- **Seuil = 3**

Pour le seuil 3, nous avons 2 nouveaux sous-ensembles par rapport au seuil 3,5 représenté dans le tableau III.8. le même outlier détecté à la position 46 a été détecté dans le scénario de test 1 avec un seuil égal 3 mais on ne remarque pas l'apparition de cet outlier avec le 2ème scénario où l'échantillon égal à 18 valeurs.

À partir de cette analyse on peut déduire qu'avec un grand nombre de données traités, des outliers peuvent être affecté par les variations du reste de données. Soient ils persistent et sont considérés comme outliers soient ils seront omis et disparaissent.

Tableau III.8: Tableau représentant les outliers détectés pour IMS avec un seuil de 3

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
8	3	1/	14	4.6466	[-1.8486,4.6466]	0.1017
18	3	2/	14	4.8009	[-34.3201, 4.8009]	-1.0293
			34	-34.3201		
19	3	3/				-1.0767

Tableau III.8: Tableau représentant les outliers détectés pour IMS avec un seuil de 3 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position	Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore	Numéro sous-ensemble
			14	4.4686	[-31.1595,4.4686]	
			34	-31.1595		
20	3	5/	35	-9.2382	[-28.6939, 11.2675]	-0.4307
			34	-28.6939		
			35	-8.5695		
			37	11.2675		
			38	11.1348		
24	3	6/	14	4.2413	[-29.5751, 11.7409]	-0.4307
			34	-29.5751		
			35	-8.7685		
			37	11.7409		
			38	11.6037		
			46	-3.1896		
25	3	5/	14	3.7915	[-26,4384, 10.4956]	-0.3415
			34	-26,4384		
			35	-7.8385		
			37	10.4956		
			38	10.3730		

- **Seuil = 4**

Pour un seuil égal à 4, on remarque la disparition de l'outlier à la position 14 dans le tableau III.9. C'est le même outlier qui disparaissent dans les autres scénaions avec un seuil de 4. La signification de ce résultat est que le score calculé pour le outlier 14 dépend de la variation de l'ensemble de données étudié.

Tableau III.9 : Tableau représentant les outliers détectés pour IMS avec un seuil de 4

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
8	4	1/	14	4.6466	[-1.8486,4.6466]	0.1017
18	4	2/	14	4.8009	[-34.3201, 4.8009]	-1.0293
			34	-34.3201		
19	4	3/	14	4.4686	[-31.1595,4.4686]	-1.0767
			34	-31.1595		

Tableau III.9 : Tableau représentant les outliers détectés pour IMS avec un seuil de 4 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
			35	-9.2382		
20	4	5/	14	4.0138	[-28.6939, 11.2675]	-0.4307
			34	-28.6939		
			35	-8.5695		
			37	11.2675		
			38	11.1348		
22	4	4/	34	-27.3489	[-27.3489, 10.7393]	-0.3909
			35	-8.1678		
			37	10.7393		
			38	10.6128		
24	4	5/	14	4.2413	[-29.5751, 11.7409]	-0.3892
			34	-29.5751		
			35	-8.7685		
			37	11.7409		
			38	11.6037		
25	4	4/	34	-26.4384	[-26.4384, 10.4956]	-0.3415
			35	-7.8385		
			37	10.4956		
			38	10.3730		

- **Scénarios concrets d'ÉTS**

Nous avons choisi le scénario de déploiement numéro 5 décrit dans le tableau 2.2 du chapitre 2 pour montrer l'impact de changement du seuil et de la valeur de l'ensemble de données étudié sur les résultats obtenues avec le modified z-score.

- **Scénario de test 1**

Dans ce scénario de test, nous avons eu 27 sous-ensembles pour un échantillon de 12 valeurs. Dans les tableaux 13, 14 et 15 nous exposons les outliers détectés à l'aide de modified z-score et nous précisons seulement les sous-ensembles où il y a eu des changements dans le nombre des outliers ou bien dans leurs positions.

- **Seuil = 3.5**

Dans la revue de littérature, le seuil 3.5 a été définis comme le score le plus fiable. Nous commençons notre analyse avec ce seuil où nous remarquons l'apparition et la disparition d'autres outliers aux files de temps. Le premier sous-ensemble où nous pouvons détecter des outliers est celui numéro 2, après les mèmes outliers sont présent jusqu'à le sous-ensemble 12. Ce dernier enregistre 4 outliers, le sous-ensemble 13 enregistre aussi 4 outliers mais dans différents intervalles de temps. Nous remarquons que l'outlier à la position 43 disparaisse et un nouveau outlier s'est identifié à la position 48. À partir de sous-ensemble 14, nous remarquons la disparition de l'outlier 1 et 43, ces deux valeurs ne réapparaissent plus dans l'ensemble de données. De même, nous remarquons l'apparition de l'outlier à la position 82 dans le sous-ensemble 22 et sa disparition au sous-ensemble 24. Ces résultats sont représentés dans le tableau III.10.

L'augmentation de nombre des valeurs permet tout fois d'alléger l'allure des variations d'où la disparition de centaines valeurs, ainsi l'apparition d'autres valeurs montre que la variation des données augmente.

Tableau III.10 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3.5

Numéro sous-ensemble	**Seuil**	**Nombre Outliers /position**		**Valeurs de mzscore**	**[Min,Max] mzscore**	**Moyenne de mzscore**
2	3.5	2	/1	4.0045	[-5.2898, 4.0045]	-0.1345
			/2	-5.2898		
12	3.5	4	/1	3.9515	[-6.4943, 4.4529]	0.0433
			/2	-6.4943		
			/43	3.5695		
			/44	4.4529		
13	3.5	4	/1	3.5464	[-5.6753, 3.9890]	-0.1459
			/2	-5.6753		
			/44	3.9890		
			/48	-4.3895		
14	3.5	3	/2	-5.1888	[-5.1888, 3.6471]	-0.1295
			/44	3.6471		
			/48	-4.0133		

Tableau III.10 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3.5 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
16	3.5	5	/2	-5.5237	[-5.5237,5.0945]	0.0295
			/44	3.8477		
			/48	-4.2769		
			/59	5.0945		
			/61	4.8697		
17	3.5	6	/2	-5.1888	[-7.8965,4.8226]	-0.0818
			/44	3.6471		
			/48	-4.0132		
			/59	4.8226		
			/61	4.6106		
			/65	-7.8965		
22	3.5	8	/2	-4.9494	[-7.5637,4.7168]	-0.0422
			/44	3.5818		
			/48	-3.8144		
			/59	4.7168		
			/61	4.5122		
			/65	-7.5637		
			/75	4.1028		
			/82	-3.5632		
24	3.5	8	/2	-4.6264	[-7.0563, 7.4714]	-0.0016
			/48	-3.5714		
			/59	4.3583		
			/61	4.1681		
			/65	-7.0563		
			/75	3.7876		
			/91	7.4714		
			/93	-4.1334		

- **Seuil = 3**

Afin de comprendre nos données et d'analyser la méthode de modified z-score dans le temps, nous avons choisis des seuils différents afin de comparer les résultats trouvés avec le seuil choisis dans la littérature.

Nous remarquons avec ce seuil l'apparition de nouveaux outliers par rapport au seuil 3.5 dans la table III.11. Aussi avec ce seuil nous trouvons que les outliers à la position 18, 43 et 47 disparaissent.

Tableau III.11 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3

Numéro sous-ensemble	**Seuil**	**Nombre Outliers /position**		**Valeurs de mzscore**	**[Min,Max] mzscore**	**Moyenne de mzscore**
2	3	2	/1	4.0045	[-5.2897, 4.0045]	-0.1345
			/2	-5.2897		
7	3	3	/1	5.1353	[-8.2779,5.1353]	-0.2370
			/2	-8.2779		
			/18	-3.2345		
8	3	2	/1	4.5278	[-7.1590,4.5278]	-0.1972
			/2	-7.1590		
12	3	4	/1	3.9514	[-6.4943,4.4528]	0.0433
			/2	-6.4943		
			/43	3.5694		
			/44	4.4528		
13	3	6	/1	3.5463	[-5.6752,3.9890]	-0.1459
			/2	-5.6752		
			/43	3.2091		
			/44	3.9890		
			/47	-3.0511		
			/48	-4.3895		
14	3	4	/1	3.2424	[-5.1888,3.6471]	-0.1295
			/2	-5.1888		
			/44	3.6471		
			/48	-4.0132		
16	3	7	/1	3.4184	[-5.5237, 5.0945]	0.0295
			/2	-5.5237		
			/43	3.0914		
			/44	3.8477		
			/48	-4.2769		
			/59	5.0945		
			/61	4.8696		
20	3	8	/1	3.0692	[-7.3968, 4.5540]	-0.0267
			/2	-4.8527		
			/44	3.4494		
			/48	-3.7482		
			/59	4.5540		

Tableau III.11 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
			/61	4.3548		
			/65	-7.3968		
			/75	3.9564		
21	3	9	/1	3.0847	[-7.3115,4.5596]	-0.0398
			/2	-4.7844		
			/44	3.4624		
			/48	-3.6872		
			/59	4.5596		
			/61	4.3617		
			/65	-7.3115		
			/75	3.9661		
			/82	-3.4444		
24	3	10	/2	-4.6263	[-7.0563, 7.4713]	-0.0016
			/44	3.3033		
			/48	-3.5713		
			/59	4.3583		
			/61	4.1681		
			/65	-7.0563		
			/75	3.7875		
			/82	-3.3379		
			/91	7.4713		
			/93	-4.1334		

- **Seuil = 4**

Avec le seuil 4, nous remarquons la disparition des quelques outliers dans le temps (voir tableau III.12). Ces outliers sont ceux à la position 1, 44, 48 et 75. Avec ce seuil nous enregistrons les nombres le plus bas en outlier.

Tableau III.12 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 4

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
2	4	2	/1	4.0045	[-5.2897,4.0045]	-0.1345
			/2	-5.2897		
12	4	2	/2	-6.4943	[-6.4943, 4.4528]	0.0433
			/44	4.4528		
13	4	2	/2	-5.6752	[-4.3895,3.9890]	-0.1459
			/48	-4.3895		
16	4	4	/2	-5.5237	[-5.5237, 5.0945]	0.0295
			/48	-4.2769		
			/59	5.0945		
			/61	4.8697		
17	4	5	/2	-5.1888	[-7.8964, 4.8226]	-0.0818
			/48	-4.0132		
			/59	4.8226		
			/61	4.6106		
			/65	-7.8964		
18	4	4	/2	-4.9525	[-7.5489,4.6476]	-0.0740
			/59	4.6476		
			/61	4.4443		
			/65	-7.5489		
22	4	5	/2	-4.9494	[-7.5637, 4.7168]	-0.0422
			/59	4.7168		
			/61	4.5122		
			/65	-7.5637		
			/75	4.1028		
24	4	6	/2	-4.6263	[-7.0563, 7.4714]	-0.0016
			/59	4.3583		
			/61	4.1680		
			/65	-7.0563		
			/91	7.4714		
			/93	-4.1334		

- **Scénario de test 2**

Dans ce scénario de test, nous avons pu extraire 18 sous-ensembles avec un échantillon de 18 valeurs. Les tableaux III.13, III.14 et III.15 représentent les valeurs aberrantes détectés dans le temps avec leurs scores correspondant pour les seuils 3.5, 3 et 4 respectivement.

- **Seuil = 3.5**

Avec un seuil de 3.5, nous avons eu 8 outliers pour l'ensemble de données. Les outliers à la position 1, 43, 44 et 82 disparaissent en augmentant l'ensemble de données. Cependant on remarque l'apparition d'autres outliers dans le temps. (voir tableau III.13)

Tableau III.13 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3.5

Numéro sous-ensemble	**Seuil**	**Nombre Outliers /position**		**Valeurs de mzscore**	**[Min,Max] mzscore**	**Moyenne de mzscore**
2	3.5	2	/1	4.9463	[-8.1689, 4.9463]	-0.4152
			/2	-8.1689		
8	3.5	4	/1	3.9514	[-6.4943, 4.4528]	0.0433
			/2	-6.4943		
			/43	3.5694		
			/44	4.4528		
9	3.5	4	/1	3.5463	[-5.6752, 3.9890]	-0.1342
			/2	-5.6752		
			/44	3.9890		
			/48	-4.3895		
10	3.5	3	/2	-5.5237	[-5.5237, 3.8477]	-0.1300
			/44	3.8477		
			/48	-4.2769		
11	3.5	5	/2	-5.5237	[-5.5237, 5.0945]	0.0505
			/44	3.8477		
			/48	-4.2769		
			/59	5.0945		
			/61	4.8696		
12	3.5	6	/2	-4.9524	[-7.5489, 4.64758]	-0.0740
			/44	3.5203		
			/48	-3.8252		
			/59	4.6475		
			/61	4.4443		
			/65	-7.5489		
13	3.5	6	/2	-4.8527	[-7.3968, 4.5540]	-0.0197
			/48	-3.7482		
			/59	4.5540		
			/61	4.3548		
			/65	-7.3968		
			/75	3.9564		

Tableau III.13 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3.5 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
15	3.5	8	/2	-5.2004	[-7.9473,4.9561]	-0.0471
			/44	3.7635		
			/48	-4.0078		
			/59	4.9561		
			/61	4.7410		
			/65	-7.9473		
			/75	4.3109		
			/82	-3.7439		
16	3.5	8	/2	-4.6264	[-7.0563,7.4713]	-0.0016
			/48	-3.5713		
			/59	4.3583		
			/61	4.1680		
			/65	-7.0563		
			/75	3.7875		
			/91	7.4713		
			/93	-4.1334		

- **Seuil = 3**

Le tableau III.14 représente les scores des outliers détectés avec le modified z-score. Dans cet ensemble de données, nous enregistrons 10 outliers, ces outliers ne sont pas les mêmes en augmentant à chaque fois le nombre des valeurs à traiter. On remarque aussi que les scores varient selon l'ensemble de données étudiés, on enregistre parfois des augmentations ou des diminutions. Par exemple pour l'outlier numéro 2, son score est égal à -8.1689 et puis ça diminue à -4.7844 et puis ça augmente à -5.2004.

Tableau III.14 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
2	3	2	/1	4.9463	[-8.1689, 4.9463]	-0.4152
			/2	-8.1689		
8	3	4	/1	3.9514	[-6.4943, 4.4528]	0.0433
			/2	-6.4943		
			/43	3.5694		
			/44	4.4528		
9	3	6	/1	3.5463	[-5.6752, 3.9890]	-0.1342
			/2	-5.6752		
			/43	3.2091		
			/44	3.9890		
			/47	-3.0510		
			/48	-4.3895		
10	3	5	/1	3.4184	[-5.5237, 3.8477]	-0.1300
			/2	-5.5237		
			/43	3.0914		
			/44	3.8477		
			/48	-4.2769		
11	3	7	/1	3.4184	[-5.5237, 5.0945]	0.0505
			/2	-5.5237		
			/43	3.0914		
			/44	3.8477		
			/48	-4.2769		
			/59	5.0945		
			/61	4.8696		
13	3	8	/1	3.0692	[,4.5540]	-0.0197
			/2	-4.8527		
			/44	3.4494		
			/48	-3.7482		
			/59	4.5540		
			/61	4.3548		
			/65	-7.3968		
			/75	3.9564		
14	3	9	/1	3.0847	[-7.3115, 4.5596]	-0.0398
			/2	-4.7844		
			/44	3.4624		
			/48	-3.6872		
			/59	4.5596		

Tableau III.14 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
			/61	4.3617		
			/65	-7.3115		
			/75	3.9660		
			/82	-3.4444		
15	3	10	/1	3.3529	[-7.9473,4.9561]	-0.0471
			/2	-5.2004		
			/43	3.0401		
			/44	3.7635		
			/48	-4.0078		
			/59	4.9561		
			/61	4.7411		
			/65	-7.9473		
			/75	4.3109		
			/82	-3.7439		
16	3	10	/2	-4.6263	[-7.0563, 7.4713]	-0.0016
			/44	3.3033		
			/48	-3.5714		
			/59	4.3583		
			/61	4.1680		
			/65	-7.0563		
			/75	3.7875		
			/82	-3.3379		
			/91	7.4713		
			/93	-4.1334		

- **Seuil = 4**

Avec un seuil égal à 4, on enregistre seulement 6 outliers. Seules les valeurs ayant un score plus que 4 sont détectés. En outre, on trouve parfois des sous-ensembles ayant le même nombre des outliers mais les valeurs et les positions des outliers changent donc on peut conclure qu'en augmentant le nombre des données étudiées, les outliers peuvent changer. (voir tableau III.15)

Tableau III.15 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 4

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
2	4	2	/1	4.9463	[-8.1689, 4.9463]	-0.4152
			/2	-8.1689		
8	4	2	/2	-6.4943	[-6.4943, 4.4528]	0.0433
			/44	4.4528		
9	4	2	/2	-5.6752	[-5.6752, 3.9890]	-0.1342
			/48	-4.3895		
11	4	4	/2	-5.5237	[-5.5237, 5.0945]	0.0505
			/48	-4.2769		
			/59	5.0945		
			/61	4.8696		
12	4	4	/2	-4.9524	[-7.5488, 4.6475]	-0.0740
			/59	4.6475		
			/61	4.4443		
			/65	-7.5488		
15	4	6	/2	-5.2004	[-7.9473,4.9561]	-0.0471
			/48	-4.0078		
			/59	4.9561		
			/61	4.7411		
			/65	-7.9473		
			/75	4.3109		
16	4	6	/2	-4.6263	[-7.0563, 7.4713]	-0.0016
			/59	4.3583		
			/61	4.1680		
			/65	-7.0563		
			/91	7.4713		
			/93	-4.1334		

- **Scénario de test 3**

Nous avons collecté 55 sous-ensembles avec un échantillon de 6 valeurs. On augmente à chaque fois le nombre des données étudiés par 6 valeurs et on enregistre les résultats tout en modifiant le seuil.

- **Seuil = 3.5**

Avec le seuil 3.5, nous détectons 8 outliers pour l'ensemble des données étudiées. Nous remarquons la disparition de l'outlier 43 dans le sous-ensemble 25 et son score devient 3.4006. Ainsi l'outlier numéro 1 devient 3.2424. Avec l'augmentation des données traitées, nous remarquons dans le tableau III.16 que le score de quelques outliers baisse et que d'autres outliers apparaissent.

Tableau III.16 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3.5

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
4	3.5	2	/1	4.0045	[-5.2898,4.0045]	-0.1345
			/2	-5.2898		
23	3.5	4	/1	4.5103	[-7.4126, 5.0826]	0.0074
			/2	-7.4126		
			/43	4.0743		
			/44	5.0826		
25	3.5	4	/1	3.7603	[-6.0761, 4.2325]	-0.0895
			/2	-6.0761		
			/44	4.2325		
			/48	-4.7046		
28	3.5	3	/2	-5.1888	[-5.1888, 3.6471]	-0.1295
			/44	3.6471		
			/48	-4.0133		
29	3.5	4	/1	3.5464	[-5.6753, 3.9890]	-0.1361
			/2	-5.6753		
			/44	3.9890		
			/48	-4.3895		
30	3.5	3	/2	-5.5237	[-5.5237, 3.8477]	-0.1300
			/44	3.8477		
			/48	-4.2769		
31	3.5	4	/2	-5.5237	[-5.5237,5.0945]	-0.0451
			/44	3.8477		
			/48	-4.2769		
			/59	5.0945		
32	3.5	5	/2	-5.5237	[-5.5237,5.0945]	0.0295
			/44	3.8477		
			/48	-4.2769		
			/59	5.0945		

Tableau III.16 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3.5 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
			/61	4.8697		
34	3.5	6	/2	-5.1888	[-7.896, 4.8226]	-0.0818
			/44	3.6471		
			/48	-4.0133		
			/59	4.8226		
			/61	4.6106		
			/65	-7.8965		
43	3.5	8	/2	-4.9494	[-7.5637, 4.7168]	-0.0373
			/44	3.5818		
			/48	-3.8144		
			/59	4.7168		
			/61	4.5122		
			/65	-7.5637		
			/75	4.1028		
			/82	-3.5632		
47	3.5	7	/2	-4.6865	[-7.1479, 7.5684]	0.0275
			/48	-3.6177		
			/59	4.4149		
			/61	4.2222		
			/65	-7.1479		
			/75	3.8368		
			/91	7.5684		
48	3.5	8	/2	-4.6263	[-7.0563,7.4713]	-0.0016
			/48	-3.5713		
			/59	4.3583		
			/61	4.1680		
			/65	-7.0563		
			/75	3.7875		
			/91	7.4713		
			/93	-4.1334		

- **Seuil = 3**

Avec le seuil 3, nous enregistrons plus d'outliers par rapport au seuil 3.5. En considérant les valeurs ayant un score entre 3 et 3.5, nous avons 10 outliers détectés pour l'ensemble de données. 11 outliers ont été relevés pour le sous-ensemble 53 représenté dans le tableau III.17.

Tableau III.17 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
4	3	2	/1	4.0045	[-5.2897, 4.0045]	-0.1345
			/2	-5.2897		
14	3	3	/1	5.1353	[-8.2779,5.1353]	-0.2370
			/2	-8.2779		
			/18	-3.2345		
15	3	2	/1	4.7636	[-7.5319, 4.7636]	-0.2309
			/2	-7.5319		
17	3	3	/1	4.8811	[-7.8111, 4.8811]	-0.2130
			/2	-7.8111		
			/18	-3.0388		
23	3	4	/1	4.5102	[-7.4127, 4.5102]	0.0074
			/2	-7.4127		
			/43	4.0743		
			/44	5.0825		
25	3	6	/1	3.7603	[-6.0761, 4.2324]	-0.0895
			/2	-6.0761		
			/43	3.4006		
			/44	4.2324		
			/47	-3.2769		
			/48	-4.7046		
28	3	4	/1	3.2424	[-5.1888,3.6471]	-0.1295
			/2	-5.1888		
			/44	3.6471		
			/48	-4.0132		
29	3	7	/1	3.5463	[-5.6752,3.9890]	-0.1361
			/2	-5.6752		
			/43	3.2091		
			/44	3.9890		
			/47	-3.0511		
			/48	-4.3895		
			/54	-3.0194		
30	3	5	/1	3.4184	[-5.5237, 3.8477]	-0.1300
			/2	-5.5237		
			/43	3.0914		
			/44	3.8477		
			/48	-4.2769		
31	3	6	/1	3.4184	[-5.5237,5.09451]	-0.0451

Tableau III.17 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
			/2	-5.5237		
			/43	3.0914		
			/44	3.8477		
			/48	-4.2769		
			/59	5.0945		
38	3	7	/1	3.1322	[-7.5488, 4.6475]	-0.0604
			/2	-4.9524		
			/44	3.5203		
			/48	-3.8252		
			/59	4.6475		
			/61	4.4443		
			/65	-7.5488		
42	3	9	/1	3.0847	[-7,3116, 4.5596]	-0.0398
			/2	-4.7845		
			/44	3.4624		
			/48	-3.6873		
			/59	4.5596		
			/61	4.3618		
			/65	-7,3116		
			/75	3.9661		
			/82	-3.4444		
48	3	10	/2	-4.6263	[-7.0563, 7.4713]	-0.0016
			/44	3.3033		
			/48	-3.5713		
			/59	4.3583		
			/61	4.1681		
			/65	-7.0563		
			/75	3.7876		
			/82	-3.3379		
			/91	7.4713		
			/93	-4.1334		
53	3	11	/1	3.0847	[-7.3116,7.7972]	-0.0451
			/2	-4.7844		
			/44	3.4624		
			/48	-3.6873		
			/59	4.5596		
			/61	4.3617		

Tableau III.17 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 3 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
			/65	-7.3116		
			/75	3.9661		
			/82	-3.4444		
			/91	7.7972		
			/93	-4.2718		
54	3	10	/2	-4.68646	[-7.1479, 7.5684]	-0.0497
			/44	3.3462		
			/48	-3.6178		
			/59	4.4149		
			/61	4.2222		
			/65	-7.1479		
			/75	3.8368		
			/82	-3.3813		
			/91	7.5684		
			/93	-4.1872		

- **Seuil = 4**

Avec le seuil 4, seulement 6 outliers ont été détectés pour tous les données observées représenté dans le tableau III.18.

Tableau III.18 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 4

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
4	4	2	/1	4.0045	[-5.2897, 4.0045]	-0.1345
			/2	-5.2897		
5	4	1	/2	-5.8659	[-5.8659, 3.8092]	-0.3442
23	4	4	/1	4.5102	[-7.4126, 5.0825]	0.0074
			/2	-7.4126		
			/43	4.0742		
			/44	5.0825		
24	4	2	/2	-6.4943	[-6.4943, 4.4528]	0.0433
			/44	4.4528		
25	4	3	/2	-6.0761	[-6.0761, 4.2324]	-0.0895

Tableau III.18 : Tableau représentant les outliers détectés pour les données de l'ÉTS avec un seuil de 4 (Suite)

Numéro sous-ensemble	Seuil	Nombre Outliers /position		Valeurs de mzscore	[Min,Max] mzscore	Moyenne de mzscore
			/44	4.2324		
			/48	-4.7046		
32	4	4	/2	-5.5237	[-5.5237, 5.0945]	0.0295
			/48	-4.2769		
			/59	5.0945		
			/61	4.8696		
34	4	5	/2	-5.1888	, 4.8226]	-0.0818
			/48	-4.0132		
			/59	4.8226		
			/61	4.6106		
			/65	-7.8964		
38	4	4	/2	-4.9525	[-7.5488, 4.6475]	-0.0604
			/59	4.6475		
			/61	4.4443		
			/65	-7.5488		
41	4	5	/2	-4.9494	[-7.5637, 4.7168]	-0.0144
			/59	4.7168		
			/61	4.5121		
			/65	-7.5637		
			/75	4.1028		
45	4	6	/2	-5.2004	[-7.9473, 4.9561]	-0.0471
			/48	-4.0078		
			/59	4.9561		
			/61	4.7410		
			/65	-7.9473		
			/75	4.3109		
47	4	5	/2	-4.6864	[-7.1479, 7.5684]	0.0275
			/59	4.4149		
			/61	4.2221		
			/65	-7.1479		
			/91	7.5684		
48	4	6	/2	-4.6263	[-7.0563, 7.4713]	-0.0016
			/59	4.3583		
			/61	4.1681		
			/65	-7.0563		
			/91	7.4713		
			/93	-4.1334		

ANNEXE IV

RÉSULTAT DE L'ALGORITHME D'ADAPTATION AVEC LE MAHALANOBIS

Dans cette annexe, nous allons représenter les différents scénarios où nous n'avons pas pu effectuer un ajustement de ressources.

1. Scénario 2

Dans ce scénario, un deuxième jeu de données IMS est considéré (voir table 2.2). La méthode de modified Z-score a identifié 3 paires d'outliers seulement dont deux peuvent être reconnues comme vrais outliers. Ces valeurs représentent une grande variation par rapport à l'ensemble de données. Ces phénomènes ont été analysés dans le chapitre 2.

En analysant la figure IV.1, nous remarquons que le Modified Z-score a détecté un 1er outlier à la position 100. La valeur de CPU à cet instant est de 36.6 et le TH est 552 m/s. Parce que ces valeurs sont petites, l'algorithme d'adaptation n'a pas signalé un besoin d'ajustement à faire malgré la chute dans la consommation de CPU et dans le TH à la position 103. Un deuxième outlier a été détecté à la position 110 indique une augmentation des valeurs des 2 métriques. À la position 113, on remarque cette augmentation. En effet, deux augmentations peuvent être relevées dans la figure IV.1. Nous nous intéressons à la deuxième augmentation où nous avons identifié un outlier. À ce moment, la métrique CPU a enregistré une valeur de 26.6 et le throughput 402 m/s. Ainsi, notre algorithme d'adaptation n'a pas signalé aucun ajustement de ressource, d'une part parce que la valeur de CPU ne dépasse pas le 40% et d'autre part parce que nous avons un seul cœur et on ne peut pas le supprimer pour la période de 103 jusqu'à 113 où le système a connu une grande chute de consommation de ressources.

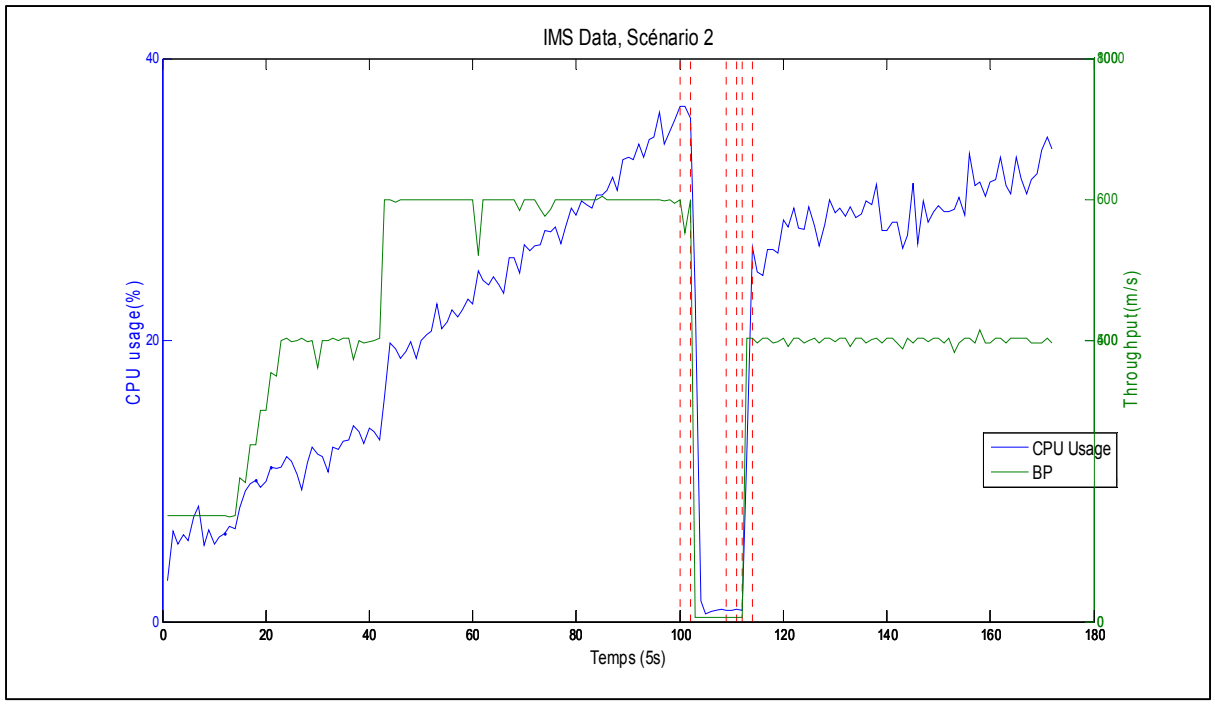

Figure IV.1 Scénario 2-Résultat de l'algorithme d'adaptation avec le Modified Z-score

La méthode Mahalanobis permet d'identifier 6 outliers tandis que le Modified Z-score a enregistré seulement 3. Le Modfied Z-score indique les variations enregistrées dans l'instant t-1 et t+1de la valeur observée, mais pas par rapport à l'ensemble des données, ceci est illustré par le 2ème outlier détecté à la position 110. Cet outlier devrait être présenté au début de la chute de données et non pas lors de la reprise du fonctionnement du système.

L'approche Mahalanobis quant à elle détecte chaque outlier qui diffère de l'ensemble de données et non pas seulement par rapport à ses voisins. Dans la figure 4.3 nous remarquons les outliers détectés pour toute la période de la chute de données. L'algorithme d'adaptation n'a pas détecté un besoin d'ajustement pour ce scénario pour les mêmes raisons listées pour le scénario 1.

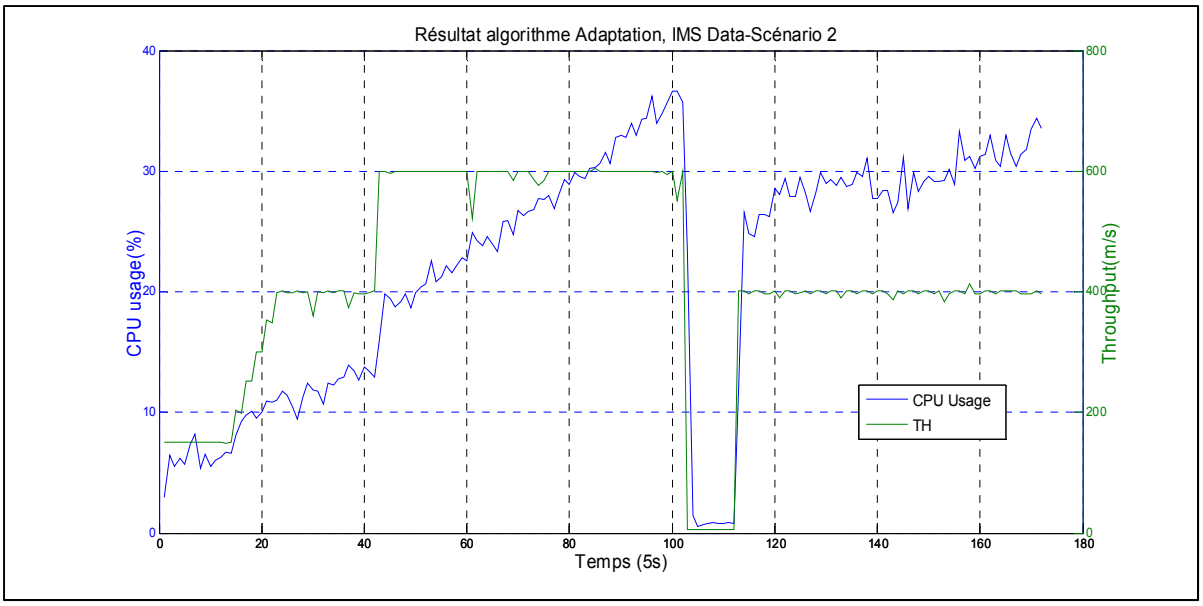

Figure IV.2 Scénario 2-Résultat de l'algorithme d'adaptation avec le Mahalanobis

2. Scénario 4

Dans ce scénario de déploiement, nous avons collecté 4 outliers avec le modified z-score et 7 avec le Mahalanobis. Nous remarquons une augmentation brusque à la position 63 et une chute de la consommation des ressources à la position 74.

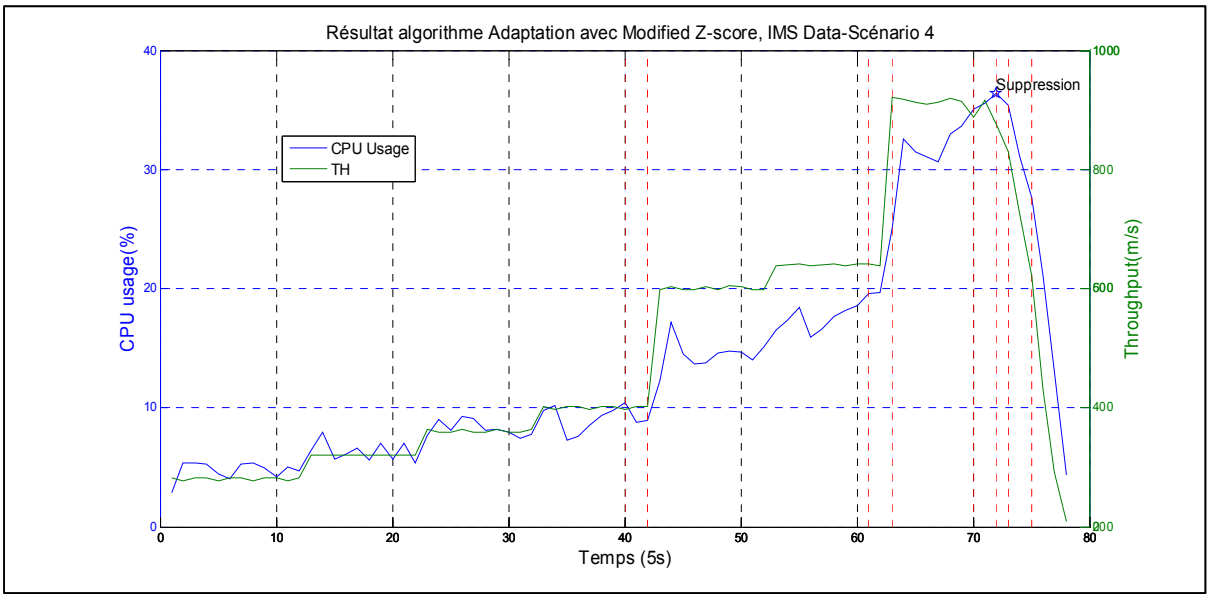

Figure IV.3 Scénario 4-Résultat de l'algorithme d'adaptation avec le modified Z-score

Les deux algorithmes que nous avons conçus nous indiquent une suppression de ressources à la position 73. La figure IV.3 indique le résultat obtenu en se basant sur le modified Z-score par contre la figure IV.4 nous montre le résultat de l'algorithme avec le Mahalanobis. L'algorithme propose une suppression d'un cœur de CPU, cependant nous avons qu'un seul cœur pour ce scénario. De ce fait, aucune adaptation ne sera effectuée pour ce système.

Avec des données pareilles pour un système possédant plus des cœurs, cette adaptation est faisable et elle évite une sous-utilisation des ressources. Cependant pour notre cas, cette adaptation n'est pas possible. Avec les filtres et les contraintes que nous avons appliqués dans l'algorithme d'adaptation, aucune adaptation ne sera effectuée.

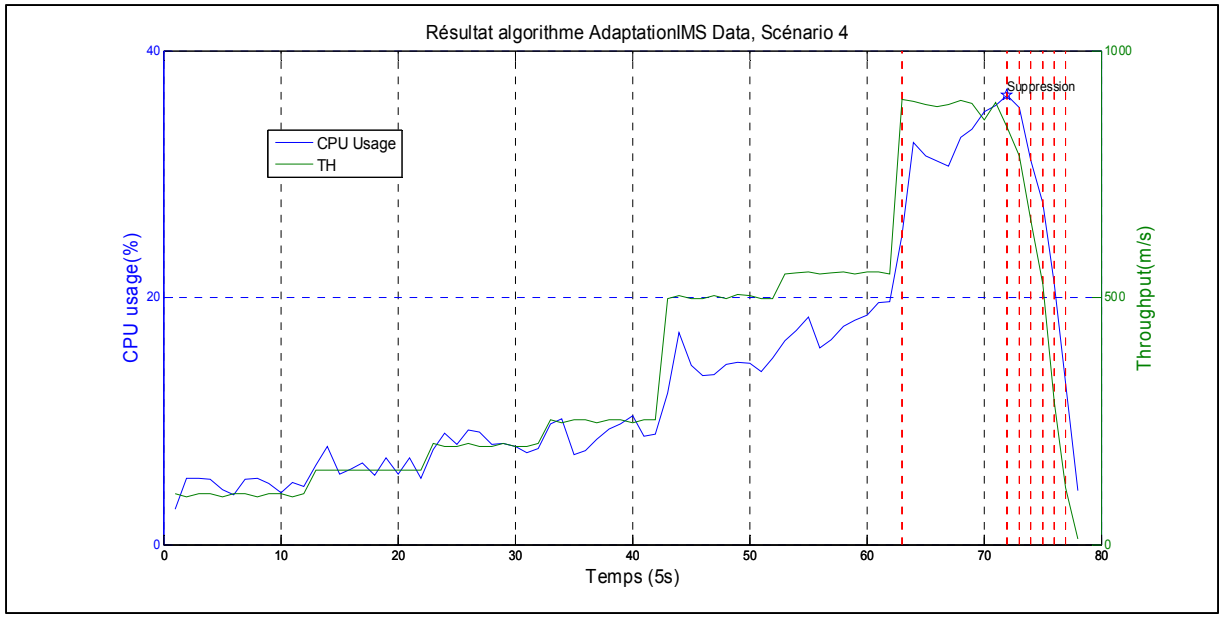

Figure IV.4 Scénario 4-Résultat de l'algorithme d'adaptation avec le Mahalanobis

3. Scénario 5

Dans ce scénario de déploiement, nous avons enregistré 5 outliers avec le modified z-score et 22 avec le Mahalanobis. Aucune adaptation n'est nécessaire pour ce scénario. La valeur maximum de CPU est de 14%. Cependant, les deux méthodes de détection d'outliers ont identifié plusieurs valeurs aberrantes. La figure IV.5 indique une grande variation dans les valeurs observées dans ce scénario pour les deux métriques. Suite à ces fluctuations nos

méthodes de détection des valeurs aberrantes identifient les valeurs qui sont différentes par rapport à l'ensemble de données.

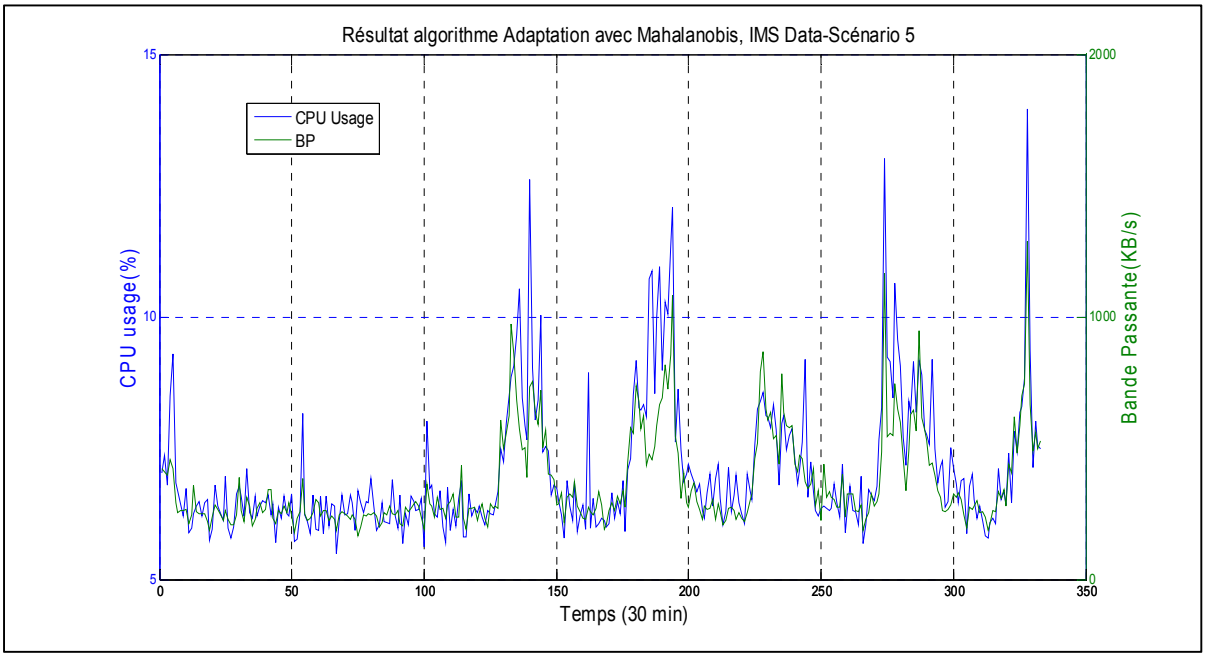

Figure IV.5 Scénario 5-Résultat de l'algorithme d'adaptation avec le Mahalanobis

4. Scénario 7

Dans ce scénario, la valeur de CPU n'est pas critique. Le maximum de l'utilisation de cpu ne dépasse pas les 30%, illustré dans la figure IV.6. L'algorithme d'adaptation de ressources n'a pas détecté des grandes variations pour les deux métriques. Aucun ajustement de ressources n'est requis dans ce cas de figure.

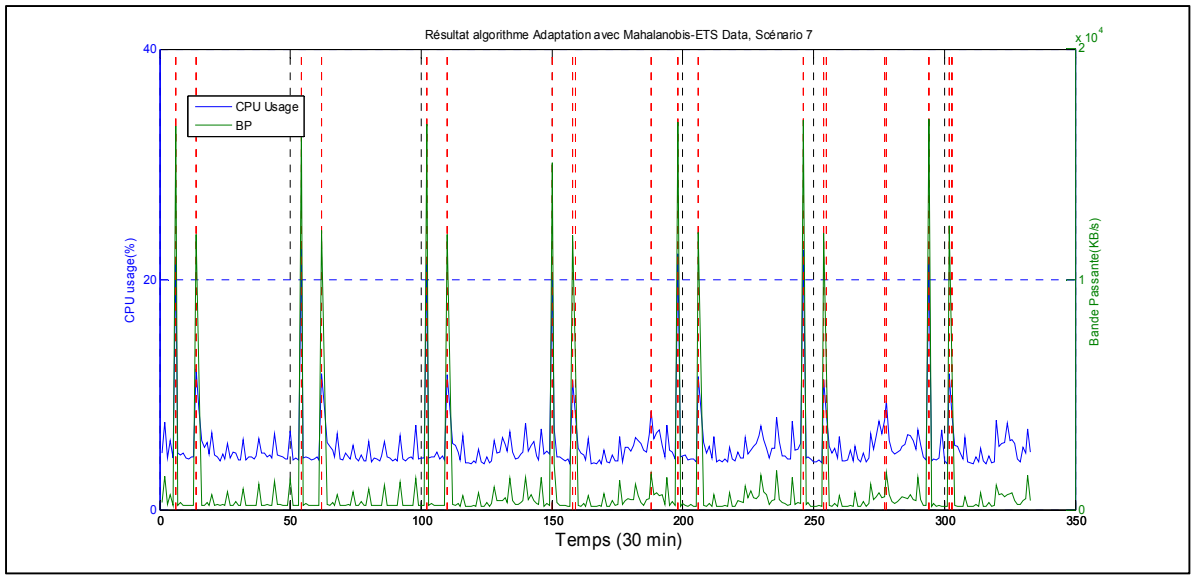

Figure IV.6 Scénario 7-Résultat de l'algorithme d'adaptation avec le Mahalanobis

5. Scénario 8

L'algorithme d'adaptation n'a pas effectué un ajustement de ressources pour la même raison que le scénario 5. Le résultat de l'algorithme ainsi que l'allure des valeurs observées sont représentés dans la figure IV.7.

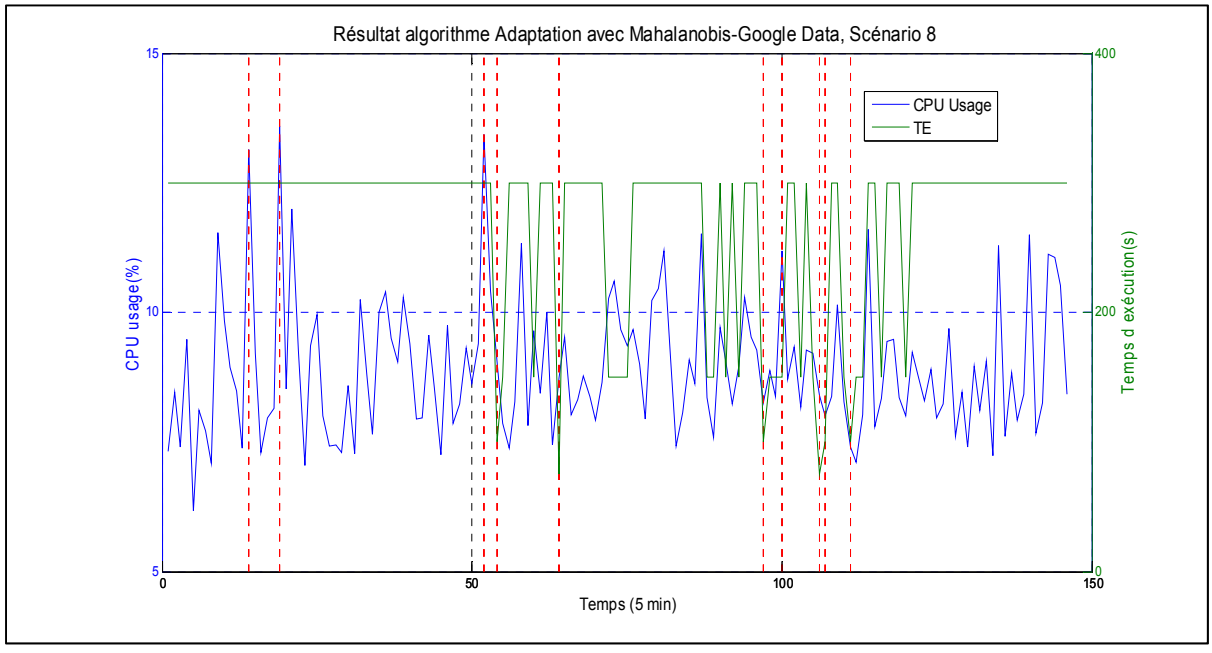

Figure IV.7 Scénario 8-Résultat de l'algorithme d'adaptation avec le Mahalanobis

6. Scénario 10

Plusieurs paramètres nous ont aidés à détecter les moments ou un système virtualisé aura besoin d'ajustement de ressources. La figure IV.8 représente les valeurs observées pour les données de google. Nous remarquons que la valeur de CPU n'a pas dépassé le 20% jusqu'à la position 140 ou il a enregistré une consommation supérieure à 30%. L'algorithme n'a pas signalé un besoin d'ajustement, simplement parce que le système a subi une diminution de ressources à l'instant qui suit cette augmentation.

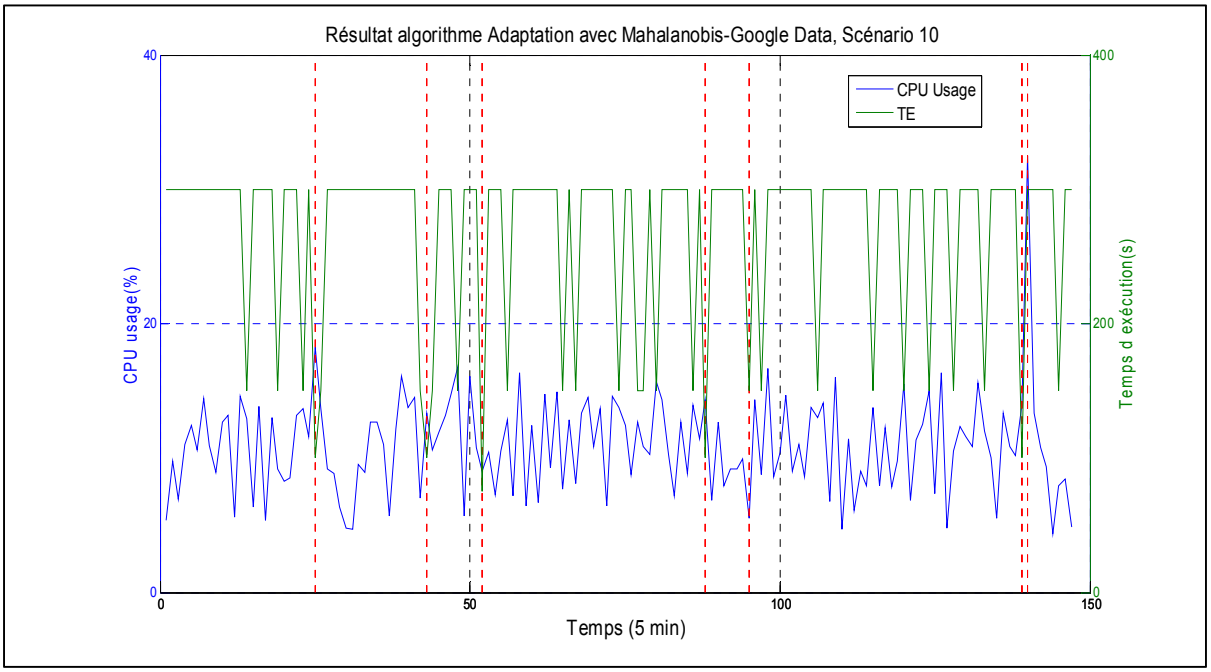

Figure IV.8 Scénario 10-Résultat de l'algorithme d'adaptation avec le Mahalanobis

BIBLIOGRAPHIE

Ahad, R., E. Chan et A. Santos. 2015. « Toward Autonomic Cloud: Automatic Anomaly Detection and Resolution ». In *2015 International Conference on Cloud and Autonomic Computing*. (21-25 Sept. 2015), p. 200-203.

Bedhiaf, I. L., O. Cherkaoui et G. Pujolle. 2009. « Performance characterization of signaling traffic in UMTS virtualized network ». In *2009 Global Information Infrastructure Symposium*. (23-26 June 2009), p. 1-8.

Been, J. M., W. S. Yang, J. H. Kim et J. O. Lee. 2015. « Management of IoT traffic using a virtualized IMS platform ». In *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*. (19-21 Aug. 2015), p. 456-459.

Calarco, G., et M. Casoni. 2013. « On the effectiveness of Linux containers for network virtualization ». *Simulation Modelling Practice and Theory,* vol. 31, p. 169-85.

Cao, L., Y. Yan, C. Kuhlman, Q. Wang, E. A. Rundensteiner et M. Eltabakh. 2017. « Multi-Tactic Distance-Based Outlier Detection ». In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. (19-22 April 2017), p. 959-970.

Carella, G., M. Corici, P. Crosta, P. Comi, T. M. Bohnert, A. A. Corici, D. Vingarzan et T. Magedanz. 2014. « Cloudified IP Multimedia Subsystem (IMS) for Network Function Virtualization (NFV)-based architectures ». In *2014 IEEE Symposium on Computers and Communications (ISCC)*. (23-26 June 2014) Vol. Workshops, p. 1-6.

Freeman, Jim. 1995. *Outliers in Statistical Data (3rd edition)*, 46.

Gong, Zhenhuan, Xiaohui Gu et John Wilkes. 2010. « Press: Predictive elastic resource scaling for cloud systems ». In *Network and Service Management (CNSM), 2010 International Conference on*. p. 9-16. Ieee.

Härdle, Wolfgang, et Léopold Simar. 2007. *Applied multivariate statistical analysis*, 22007. Springer.

Iglewicz, Boris, et David C Hoaglin. 1993. « How to Detect and Handle Outliers, ASQC basic references in quality control ». *Milwaukee, WI: American Society for Quality Control*.

Inc, Books x, et Bruce Ratner. 2012. *Statistical and machine-learning data mining : techniques for better predictive modeling and analysis of big data, second edition* (2012), 2nd ed. Coll. « Techniques for better predictive modeling and analysis of big data ». Boca Raton, Fla.: CRC Press, 1 ressource en ligne. p.

Jiang, H., A. Iyengar, E. Nahum, W. Segmuller, A. N. Tantawi et C. P. Wright. 2012. « Design, Implementation, and Performance of a Load Balancer for SIP Server Clusters ». *IEEE/ACM Transactions on Networking,* vol. 20, nº 4, p. 1190-1202.

Jiang, H., A. Iyengar, E. Nahum, W. Segmuller, A. Tantawi et C. P. Wright. 2009. « Load Balancing for SIP Server Clusters ». In *INFOCOM 2009, IEEE*. (19-25 April 2009), p. 2286-2294.

Jing, Sun, Tian Ruixiong, Hu Jinfeng et Yang Bo. 2009. « Rate-based SIP flow management for SLA satisfaction ». In *2009 IFIP/IEEE International Symposium on Integrated Network Management*. (1-5 June 2009), p. 125-128.

Johnson, Richard Arnold, et Dean W. Wichern. 2007. *Applied multivariate statistical analysis* (2007), 6e éd. Upper Saddle River, N.J.: Pearson/Prentice Hall, xviii, 773 p. p.

Kannan, K Senthamarai, K Manoj et S Arumugam. 2015. « Labeling Methods for Identifying Outliers ». *International Journal of Statistics and Systems,* vol. 10, nº 2, p. 231-238.

Liao, Lingxia, Victor C. M. Leung et Min Chen. 2015. « Virtualizing IMS core and its performance analysis ». In *5th International Conference on Cloud Computing, CloudComp 2014, October 19, 2014 - October 21, 2014*. (Guilin, China) Vol. 142, p. 53-65. Coll. « Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST »: Springer Verlag. < http://dx.doi.org/10.1007/978-3-319-16050-4_5 >.

Lorido-Botran, Tania, Sergio Huerta, Luis Tomás, Johan Tordsson et Borja Sanz. 2017. « An unsupervised approach to online noisy-neighbor detection in cloud data centers ». *Expert Systems with Applications,* vol. 89, p. 188-204.

Lu, F., H. Pan, X. Lei, X. Liao et H. Jin. 2013. « A Virtualization-Based Cloud Infrastructure for IMS Core Network ». In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. (2-5 Dec. 2013) Vol. 1, p. 25-32.

Maciá-Pérez, Francisco, Jose Vicente Berna-Martinez, Alberto Fernández Oliva et Miguel Alfonso Abreu Ortega. 2015. « Algorithm for the detection of outliers based on the theory of rough sets ». *Decision support systems,* vol. 75, p. 63-75.

Mahalanobis, P.C. 1936. « On the Generalized Distance in Statistics ». *Proceedings of the National Institute of Science of India,* vol. 2, p. 49-55.

Maheshwari, K., et M. Singh. 2016. « Outlier detection using divide-and-conquer strategy in density based clustering ». In *2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*. (23-25 Dec. 2016), p. 1-5.

Montazerolghaem, A., M. H. Yaghmaee, A. Leon-Garcia, M. Naghibzadeh et F. Tashtarian. 2016. « A Load-Balanced Call Admission Controller for IMS Cloud Computing ». *IEEE Transactions on Network and Service Management,* vol. 13, nº 4, p. 806-822.

Sinha, S. K. 1979. « Outliers in Statistical Data (Vic Barnett and Toby Lewis) ». *SIAM Review,* vol. 21, nº 4, p. 576-577.

Sun, J., J. Hu, R. Tian et B. Yang. 2007. « Flow Management for SIP Application Servers ». In *2007 IEEE International Conference on Communications*. (24-28 June 2007), p. 646-652.

Tan, Huailiang, Lianjun Huang, Zaihong He, Youyou Lu et Xubin He. 2014. « DMVL: An I/O bandwidth dynamic allocation method for virtual networks ». *Journal of Network and Computer Applications,* vol. 39, p. 104-116.

Wang, Bo-Chun, Y Tay et Leana Golubchik. 2012. « Resource allocation for network virtualization through users and network interaction analysis ».

Wei, Zhang, Lei Weimin, Chen Xiao et Liu Shaowei. 2013. « Architecture and Key Issues of IMS-based Cloud Computing ». In *2013 IEEE 6th International Conference on Cloud Computing (CLOUD), 28 June-3 July 2013*. (Los Alamitos, CA, USA), p. 629-35. Coll. « 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD) »: IEEE Computer Society. < http://dx.doi.org/10.1109/CLOUD.2013.34 >.

Wenzhi, Liu, Li Shuai, Xiang Yang et Tang Xiongyan. 2012. « Dynamically adaptive bandwidth allocation in network virtualization environment ». *Advances in Information Sciences and Service Sciences,* vol. 4, nº 1.

Wilkes, John. 2011. « Google cluster data ». In *Google research blog, Nov.* < https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md >.

Yuen, Ka-Veng, et He-Qing Mu. 2012. *A novel probabilistic method for robust parametric identification and outlier detection*, 30.