Caractérisation de la propagation des anomalies dans un environnement conteneurisé via une approche bayésienne adaptative

par

Soufiene BEN DAHSEN

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 16 JUILLET 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Abdelouahed Gherbi, Directeur de mémoire
Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

M. Chamseddine Talhi, Président du jury
Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

M. Sègla Jean-Luc Kpodjedo, Membre du jury
Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

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

LE 13 JUILLET 2021

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Au terme de ce travail, je saisis cette occasion pour exprimer mes vifs remerciements à ma famille, mes amis et à toute personne ayant contribué, de près ou de loin, à la réalisation de cette maîtrise.

Qu'il me soit permis d'exprimer toute ma gratitude à mon professeur M. Abdelouahed Gherbi pour ses conseils judicieux, son assistance et ses remarques précieuses qui m'ont permis de mener efficacement ce travail.

J'adresse mes remerciements à M. Ahmed Bali, qui n'a pas cessé de me diriger et conseiller. Son savoir et ses qualités m'ont permis de mener à bien mon travail.

En fin, je suis reconnaissant au support financier reçu conjointement de la part l'ETS et de Mitacs.

Caractérisation de la propagation des anomalies dans un environnement conteneurisé via une approche bayésienne adaptative

Soufiene BEN DAHSEN

RÉSUMÉ

Une architecture basée sur des conteneurs est une approche prometteuse pour la construction et le déploiement de systèmes distribués. L'expansion en matière de complexité de ces infrastructures a compliqué le processus de surveillance.

En raison de la prolifération des objets et des mesures à suivre, la tâche de surveillance présente de sérieuses lacunes pour détecter une déviation de performance sur un seul conteneur et ses effets sur les autres conteneurs dépendants. Par conséquent, agir de manière prospective pendant la tâche de surveillance en prédisant l'état du système permet de contrôler les ressources du système et aide les outils d'orchestration à réagir de manière proactive en cas d'anomalies. En effet, la prédiction des anomalies est une option prometteuse pour aider à suivre le comportement anormal du système global.

Ce travail étudie une approche d'apprentissage bayésienne adaptée, qui se base sur des mesures de suivi des niveaux d'utilisation des ressources pour détecter et prédire la propagation d'un comportement anormal dans une architecture distribué composée de nœuds dans lesquels différentes applications conteneurisées sont déployées. Notre évaluation montre un taux de précision de prédiction prometteur.

Mots-clés: conteneur, cluster, apprentissage bayésien, prédiction d'anomalie

Adaptive Bayesian Approach to Anomaly Propagation Characterization in Clustered Computing Environments

Soufiene BEN DAHSEN

ABSTRACT

A container-based architecture is a promising approach for building and deploying distributed systems. The expansion in terms of complexity of such infrastructures has complicated the monitoring process.

Due to the proliferation of objects and metrics to track, monitoring task face serious gaps to catch a performance drop on a single container and its effects on other dependent containers. Therefore, acting prospectively during the monitoring task by analyzing the system state helps controlling system resources and assists orchestration tools to react proactively in case of workloads anomalies. In fact, anomaly prediction is a promising option to help in tracking anomalous behavior of the overall system.

This work investigates, based on tracking resources metrics, how to predict anomalous workload behavior in a cluster architecture consisting of nodes in which different containerized application are deployed. It provides an adapted Bayesian learning approach that predicts anomalous behavior in a containerized cluster environment based on the observed metrics. Our evaluation shows a promising prediction accuracy.

Keywords: container, cluster, bayesian learning, anomaly prediction

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CPD	Conditional probability distribution
CPU	Central processing unit
ETS	École de Technologie Supérieure
MAPE-K	Monitor, Analyse, Plan, Execute, Knowledge
RMSE	Root mean square error
SUT	System Under Test
VM	Virtual machine

INTRODUCTION

0.1 Contexte et Motivations

La conteneurisation des architectures apparaît comme une approche émergente pour construire et déployer des systèmes d'applications distribuées. La charge supportée par chaque conteneur administre la décision de mise en échelle des applications. La gestion de cette charge se complexifie avec l'expansion en échelle des infrastructures distribuées. En effet, dans ce cadre, la dégradation en matière de performance survenue sur un conteneur se propagerait en cascade et affecterait la performance de ses dépendants. Cette dégradation est observée comme une charge de travail anormale, qui pourrait potentiellement engendrer une défaillance.

Par conséquent le contrôle et l'analyse préalable des ressources d'un système distribué représentent une piste solide pour prédire les caractéristiques de la charge de travail des éléments de l'infrastructure en question (conteneur, nœud, cluster), ce qui permet de prévenir en premier lieu l'apparition d'une charge anormale, et de contribuer en second lieu à l'amélioration de la performance globale du système.

Par ailleurs, dans le cadre d'une infrastructure infonuagique, l'hétérogénéité des ressources, entraînant des variations des charges, nuit directement à la performance de l'ensemble du système. La modélisation du comportement des applications en infonuagique fait face donc à un défi de complexité croissante et requiert généralement une connaissance préalable des caractéristiques de la charge de travail des éléments.

Ainsi, la détection et la prédiction des anomalies pourront présenter des optiques prometteuses pour retracer un comportement anormal. Notre objectif est de capturer un comportement anormal des ressources dans une architecture conteneurisée (constituée de nœuds dont les applications sont déployées sur des conteneurs) en investiguant une séquence d'observations émises par chaque ressource.

0.2 Problématique

L'expansion des systèmes distribués actuels complique le processus de détection des performances du système et d'identification des causes racines de ses défaillances. La virtualisation semblait un moyen efficace pour optimiser la consommation des ressources dans un environnement distribué. Toutefois la surcharge induite par le système d'exploitation impacte intrinsèquement la performance globale de l'environnement en question. Par conséquent, pour pallier cette contrainte, les conteneurs proposaient un autre niveau d'abstraction isolant les processus au niveau du système d'exploitation afin d'éviter les surcharges.

Les principales hypothèses de ce travail sont les suivantes :

– il existe une corrélation positive entre la valeur de la charge gérée par un système et les métriques observées au cours de son exécution. Cette valeur peut être utilisée comme indicatrice d'anomalie.
– un seul élément ou plus d'une architecture distribuée pevent être la cause d'une défaillance, et cette dernière peut impliquer la propagation d'un comportement anormal vers d'autres éléments de l'architecture.

Ces hypothèses mèneront à la formulation de notre problématique comme suit : l'analyse proactive du comportement anormal dans un système distribué et la mise en place de multiples actions de prévention pour des anomalies prédites permettent non seulement d'affirmer la fiabilité et la robustesse du système, mais aussi de réduire ses coûts et ses délais d'exploitation et maintenance.

Ce travail fait face à des défis que nous traitons à partir de perspectives multiples : détection, prédiction et récupération des anomalies, contexte et contraintes de l'architecture et recueil de données à partir d'un environnement conteneurisé.

0.3 Objectifs et contributions

L'objectif de ce mémoire est de proposer une approche adaptative, capable de détecter, identifier et prédire, à partir des observations des ressources consommées, un comportement anormal dans un environnement hiérarchique de conteneurs et nœuds. Pour y parvenir, nous analysons la performance des conteneurs et des nœuds en leur injectant différentes anomalies pour recueillir davantage de données sur la performance, et nous générons une charge de travail synthétique pour simuler les demandes réelles d'utilisation.

Les phases de détection, d'identification et adaptation sont alignés avec la boucle de contrôle autonomique MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge).

Ce qui suit reflète les contributions de ce mémoire :

– identifier un comportement anormal en faisant correspondre la dégradation observée à sa composante anormale sous-jacente.
– repérer les conséquences potentielles du comportement anormal
– appliquer des actions de récupération proactives pour recouvrir l'anomalie identifiée.

Pour y parvenir, nous proposons une approche bayésienne adaptative qui modélise le mécanisme de gestion des défaillances dans une architecture distribuée. Ce mécanisme comporte une phase d'analyse, de prédiction et identification d'anomalies en se basant sur les observations émises par chaque ressource de l'architecture et une phase d'adaptation mettant à jour les paramètres de la modélisation de l'architecture et évaluant des actions de récupération. L'approche proposée est évaluée pour sa capacité à détecter, prédire et corriger des comportements anormaux des composants d'un système distribué.

0.4 Organisation du mémoire

Le chapitre 1 illustre les principaux concepts rencontrés dans ce mémoire. Le chapitre 2 expose une revue de littérature comparant le travail proposé par rapport à d'autres approches existantes. Le chapitre 3 introduit l'infrastructure adaptative considérée, capable de détecter et prédire des anomalies, et détaille ses différents volets. L'interaction entre le volet de gestion des anomalies et le volet d'adaptation est automatisé par l'adoption de la boucle de contrôle autonomique MAPE-K pour signaler les anomalies identifiées par le premier volet, et les acquitter au processus d'adaptation et récupération en second volet. Le chapitre 4 précise le protocole expérimental adopté pour démontrer la capacité de l'architecture adaptative proposée à détecter, prédire et récupérer des anomalies de performance dans une infrastructure infonuagique distribuée. Pour ce faire, différents scénarios d'opérations sont générés et diverses anomalies sont injectées dans des compartiments de l'infrastructure afin d'évaluer et vérifier la précision de l'approche. Nous concluons le travail en discutant des résultats de l'approche bayésienne adaptative et en mettant en évidence les orientations des travaux futurs pour améliorer notre proposition.

CHAPITRE 1

CONCEPTS GÉNÉRAUX

1.1 Introduction

Ce chapitre illustre les principaux concepts rencontrés dans ce mémoire, à savoir les systèmes adaptatifs, la gestion des architectures distribuées en clusters et conteneurs, les différentes anomalies de performance, l'analyse des causes racines, et les modèles bayésiens.

1.2 Systèmes adaptatifs

Un système adaptatif est un système qui réagit en fonction d'un éventuel changement de son environnement. Ce système suit un mécanisme fermé d'étapes répondant à la chaîne de contrôle MAPE-K. Ces étapes sont :

- étape de surveillance : collecter les données à partir des ressources administrées. Ces ressources comprennent des conteneurs, des services, des serveurs, des machines virtuelles ou des applications logicielles.
- étape d'analyse : analyser les variations significatives d'un système provenant de la phase de contrôle.
- étape de planification : fournir les mesures requises pour atteindre les objectifs du système.
- étape d'exécution : appliquer les mesures spécifiées en phase de planification pour améliorer le comportement des composantes contrôlé.
- knowledge : est un entrepôt de donnés intégrant des métriques et des règles de décision. Ces données sont partagées entre les étapes de surveillance, d'analyse et de planification.

Un système adaptatif supervise en permanence les changements et actualise ses propres modèles, qui sont analysés pour repérer toute dérogation aux obligations. Pour ce faire, plusieurs mécanismes d'analyse sont appliqués au moment de l'exécution, et des stratégies de redressement

s'appliquent également pour traiter les défaillances. Par exemple, des modèles bayésiens sont adoptés pour prédire la fiabilité et la performance d'un système.

Un système adaptatif adresse les variations en cours d'exécution relatives aux exigences fonctionnelles et non fonctionnelles. Les spécifications fonctionnelles décrivent la fonctionnalité d'un système (par exemple, l'envoi de données d'un point A à un point B). Les spécifications non fonctionnelles décrivent généralement la qualité de la fonctionnalité du système (par exemple, la consommation des ressources). Notre projet se focalise sur les exigences non fonctionnelles.

1.3 Charge de travail dans une infrastructure conteneurisée

Un nœud est une machine virtuelle (VM) ou physique qui appartient généralement à un cluster. Il comprend un ou plusieurs conteneurs. Chaque conteneur exécute un ensemble de tâches. Les nœuds et les conteneurs opèrent sous cluster qui exécute ces nœuds sur une machine locale ou dans un environnement infonuagique.

Les conteneurs d'un nœud sont issus d'une image de conteneur, qui est un paquet exécutable et portable d'un logiciel. Pour planifier et orchestrer les conteneurs, un agent est installé au niveau du nœud. Chaque nœud d'un cluster dispose des informations relatives à son adresse, à son état ou à sa disponibilité. Un nœud fournit des informations sur la disponibilité de ses ressources (par exemple, consommation CPU ou mémoire, nombre maximum de conteneurs qui peuvent être ordonnancés en nœud). La capacité d'un nœud est répartie de manière égale ou inégale entre ses conteneurs. Pour y parvenir, il existe des outils de gestion de clusters de conteneurs tels que Kubernetes [1] et Docker Swarm [2].

Dans un environnement hétérogène et dynamique de clusters et conteneurs, la supervision de la charge de travail est corrélée au suivi des ressources allouées. En fait, la charge de travail qui se cumule au fil du temps peut provoquer une saturation des ressources. Le système peut générer une charge imprévisible et fluctuante dans le temps qui sollicite différentes ressources (CPU,

[1] https://kubernetes.io/fr/docs/concepts/overview/what-is-kubernetes/

[2] https://docs.docker.com/engine/swarm/

mémoire, disque) à divers intervalles. Les répercussions indésirables d'une telle charge de travail occasionnent des anomalies logicielles associées à des coûts importants. Les explications de ces dysfonctionnements sont attribuées à la configuration des logiciels, aux ressources restreintes des machines virtuelles, aux mises à jour logicielles, à la maintenance du système, voire à des intrusions en sécurité. Ainsi, l'apparition fréquente d'anomalies entraîne des défaillances qui sont difficiles à détecter dans un environnement dynamique.

1.4 Anomalies de performance

Une anomalie est une déviation du comportement d'un système dégradant ses performances. Malkowski, Hedwig & Pu (2009) qualifient l'anomalie comme une cause racine d'un comportement inhabituel causée par une occupation excessive de certaines ressources du système. En outre, d'autres publications, notamment Jung, Swint, Parekh, Pu & Sahai (2006), soulignent que certaines applications ou métriques du système sont en corrélation avec une dégradation observée des performances. Ces mesures sont appelées indicateurs d'anomalie.

Deux démarches courantes ont été poursuivies dans les études sur la détection des anomalies de performance d'une infrastructure logicielle :

– détection d'anomalies basées sur des spécifications de performance pour évaluer les capacités d'un système à satisfaire des contraintes précises (Ibidunmoye, Hernández-Rodriguez & Elmroth, 2015; Becker, Luckey & Becker, 2013).
– détecter les anomalies à partir d'une déviation de performance. Cette démarche est suivie quand aucune spécification de performance ne se présente, mais que la performance s'écarte des conditions ordinaires. Pour identifier une déviation ou anomalie de performance, deux voies sont envisageables, soit définir des seuils, soit disposer d'une expertise sur le système.

Nous suivons dans le cadre de notre projet le principe de détecter des anomalies en se basant sur la dérivation des comportements ordinaires pour les motifs suivants :

– ce principe est plus flexible que l'approche des spécifications de performance car l'environnement est dynamique et les spécifications opérationnelles varient fréquemment.
– dans un environnement dynamique et distribué, la dégradation des performances se manifeste à différents niveaux, notamment au niveau logiciel (par exemple une mauvaise configuration), au niveau matériel (par exemple la latence du réseau) ou au niveau infrastructure (par exemple la défaillance d'un conteneur).

1.4.1 Catégories d'anomalies

Les anomalies de performance se manifestent de différentes manières selon les applications et les systèmes :

– anomalie à occurrence unique : une anomalie témoigne une saturation prédominante d'une seule ressource (CPU, mémoire, disque, E/S réseau) au niveau d'un constituant (conteneur, nœud ou cluster).
– Anomalie à occurrences multiples : les ressources de différents éléments du système sont susceptibles de se saturer simultanément en raison de l'interdépendance de ces éléments.

Le traitement de ces anomalies de comportement permet d'éviter leur réapparition. Avizienis, Laprie & Randell (2001) proposent trois phases pour traiter les anomalies :

– diagnostiquer une anomalie : sert à identifier et à recueillir les causes des dysfonctionnements, en matière de localisation et de type.
– isoler l'anomalie : vise à exclure les éléments défectueux de toute implication future dans la prestation de services.
– reconfiguration du système : réaffecter les fonctions entre les éléments intacts.

1.4.2 Analyse des causes racines

Les causes racine sont des démarches visant à spécifier les origines des défauts observées. Dans un contexte de performance, c'est le moyen de détecter les causes racines des anomalies de fonctionnement. Chaque cause racine peut provoquer une ou plusieurs défaillances, et chaque défaillance peut être associée à une ou plusieurs causes racines.

Par exemple, une défaillance observant une forte latence peut découler d'une cause racine décrivant une ressource surchargée. Ainsi, pour remonter aux causes racines qui provoquent certaines figures de défaillance, une analyse de ses causes racines est indispensable pour repérer et isoler les anomalies de fonctionnement.

Cependant, pour analyser les causes racines de la performance d'un système, des connaissances d'experts sont requises pour préciser la configuration à établir pour repérer les causes d'un problème, ou de se procurer du code source du système en vue de retracer et de diagnostiquer les anomalies à l'origine des défaillances.

À la différence des contraintes mentionnées précédemment, notre travail propose une architecture adaptative qui détecte et prédit des anomalies dans une infrastructure distribuée. En fonction des défauts identifiés, notre approche applique des mesures correctives pour affiner ultérieurement les processus de détection et prédiction des anomalies.

1.5 Dépendances d'anomalies

Les travaux de Ghaith, Wang, Perry, Jiang, O'Sullivan & Murphy (2016), Farshchi, Schneider, Weber & Grundy (2016) et Wert (2015) ont examiné les anomalies en contexte infonuagique suivant deux directions principales :

– caractérisation d'anomalie : une anomalie est qualifiée au moyen de méthodes statistiques et analytiques qui ciblent l'utilisation des ressources (par exemple CPU, mémoire). Cette caractérisation comprend détection, identification de la cause racine de l'anomalie et le rétablissement de cette dernière. Les métriques ainsi que la charge de travail fournissent un

indicateur de base pour qualifier le comportement d'une anomalie. Dans une telle situation, les fichiers log comprenant des mesures de métriques sont examinés pour capturer l'état dynamique de la charge de travail des composants du système. Les travaux de (Samir & Pahl, 2019a) et (Samir & Pahl, 2019c) ont enquêté la correspondance entre la charge de travail et la consommation de ressources, et ils nous ont démontré une forte corrélation positive entre ces deux domaines.

– prédiction d'anomalie : la prédiction est effectuée en analysant l'utilisation des ressources des composants. Dans un environnement distribué de conteneurs, différentes méthodes sont destinées à analyser le comportement d'une anomalie et à prédire son comportement futur (Samir & Pahl, 2019b).

En général, les composants sont susceptibles de rencontrer plusieurs types d'anomalies :

– anomalie permanente : elle s'écarte de son comportement normal et demeure présente jusqu'à ce qu'une action corrective soit menée. Par exemple, un nœud cesse de fonctionner jusqu'à ce qu'il soit réparé (Iber, Rauter, Krisper & Kreiner, 2017).
– anomalie transitoire : elle demeure active pendant une courte durée. Cette variante est souvent repérée par les erreurs qui découlent de sa propagation (Ibidunmoye *et al.*, 2015). Elle est considérée comme un symptôme de dégradation de performances et ne requiert pas nécessairement une restauration du système.
– anomalie temporaire : cette anomalie persiste pendant une durée déterminée, puis le composant retrouve son comportement normal (Syer, Shang, Jiang & Hassan, 2017). Elle se manifeste comme un symptôme de dégradation et requiert une intervention corrective. Par exemple, si deux conteneurs C1 et C2 sont dépendants, lorsque C2 est surchargé, le C1 est sous-chargé, et dès que C2 retrouve sa charge normale, C1 fonctionnera normalement.

En plus, la saturation des ressources d'un composant est une cause majeure de l'observation des dysfonctionnements. Par exemple, les CPU ou les mémoires sont surutilisés dès qu'ils dépassent

des seuils de 80% de consommation. La congestion du réseau se manifeste en raison de la suroccupation de la bande passante, ce qui ralentit ou interrompt le flux de trafic.

Le modèle adaptatif proposé applique une gestion des défaillances au niveau des conteneurs et des nœuds. Il assume les informations préalables concernant :

– la dépendance entre les conteneurs, les nœuds et les clusters.
– l'utilisation des ressources par les conteneurs, les nœuds et les clusters.

Nous admettons que l'apparition d'un comportement erroné au niveau d'un composant peut provoquer une anomalie au niveau d'un autre composant s'il existe une dépendance entre eux. Une telle faute peut se propager horizontalement sur le plan du conteneur ou du nœud, ou peut s'étendre verticalement à différents niveaux du système. Nous nous intéressons plutôt à la gestion de la défaillance enregistrée sur des conteneurs, des nœuds et des clusters.

À la lumière de la conception de notre réseau et des travaux de Kratzke (2017) et Raj & Raman (2018), nous distinguons différents profils de dépendance des composants illustrant l'interaction entre les éléments du système.

1.5.1 Dépendance Conteneur-conteneur

Ça concerne une ou plusieurs anomalies qui se manifestent au niveau d'un ou plusieurs conteneurs.

– auto-dépendance : elle implique qu'un ou plusieurs conteneurs sont saturés. Le conteneur surchargé n'a aucune dépendance vis-à-vis d'autres conteneurs.
– interdépendance : elle implique que deux ou plusieurs conteneurs dépendants du même nœud subissent une importante surcharge. Dans ce cas, un conteneur saturé affecte considérablement ses dépendants.
– dépendance externe : elle implique que les conteneurs dépendant d'un nœud subissent une surcharge qui affecte implicitement les autres conteneurs dépendant d'un autre nœud.

1.5.2 Dépendance nœud-Conteneur

Ça correspond à une dépendance parent-enfant. Dans une telle situation, une anomalie se manifeste au niveau du conteneur ou du nœud. Au niveau du conteneur, les conteneurs épuisent les ressources d'un nœud, ce qui provoque sa surcharge. Quant au nœud, s'il ne dispose pas de suffisamment des ressources, une surcharge apparaît à la fois sur le conteneur et sur le nœud.

1.5.3 Dépendance nœud-nœud

Ça concerne la dépendance sur le plan des nœuds. Nous distinguons deux sortes de dépendance :

- auto-dépendance : elle implique qu'un ou plusieurs nœuds isolés (c'est-à-dire qu'un nœud ne possède aucune relation avec d'autres nœuds) affichent une surcharge. Dans un tel cas, les nœuds saturés ne nuisent pas aux autres nœuds du cluster.
- interdépendance : elle implique que les nœuds d'un même cluster subissent une surcharge en raison de leur dépendance mutuelle. Dans un tel cas, un nœud anormal impacte directement le ou les autres nœuds dépendants.

1.6 Réseaux bayésiens

Les réseaux bayésiens représentent une modélisation graphique probabiliste servant à construire des modèles à partir de données et / ou d'opinions d'experts. Ils sont utiles pour satisfaire à plusieurs applications d'analyses, notamment descriptives (modélisation, simulation), de diagnostic (raisonnement, dépannage), prédictives (détection d'anomalies, apprentissage supervisé, non supervisé) et prescriptives (support de décision).

Dans un réseau, les nœuds représentent des variables discrètes ou continues, et les liens indiquent la dépendance entre nœuds. Cela permet de visualiser facilement la relation entre les variables. Toutefois, l'absence d'un lien entre deux nœuds ne signifie pas qu'ils sont complètement

indépendants, car ils peuvent être reliés via d'autres nœuds. Ils peuvent cependant devenir dépendants ou indépendants en fonction de l'évidence appliquée sur les autres nœuds.

1.6.1 Théorie bayésienne

Un réseau bayésien est un graphe orienté acyclique où chaque nœud est associé à des informations probabilistes quantitatives. Chaque nœud est rattaché à une variable aléatoire et à des liens (arcs) le reliant aux autres nœuds. Un lien partant du nœud X vers le nœud Y signifie que X est le parent de Y et implique une relation de dépendance conditionnelle entre eux. Chaque nœud Ai se caractérise par une distribution de probabilité conditionnelle exprimée en équation suivante :

$$P(A_i|parents(A_i)) \tag{1.1}$$

L'ensemble des nœuds et liens forme la topologie du réseau et illustre les relations de dépendance et indépendances existant dans le réseau. Chaque lien partant du nœud X vers le nœud Y montre que la variable X affecte la variable Y. Concrètement, la valeur que prend la variable X affecte la distribution de probabilité de la variable Y. Si un nœud du graphe n'a pas de parents, il détient simplement une distribution de probabilité propre à lui. Par ailleurs, si ce nœud X a n parents, il dispose d'une distribution de probabilité conditionnelle pour chaque parent Y_i. Cela signifie que, pour chaque parent Y_i, nous disposons d'un tableau de distribution de probabilités représenté par $P(X|Yi)$.

Une fois que le graphe des dépendances est établi, nous spécifions les distributions de probabilité conjointes pour les ensembles de nœuds dépendants. Le théorème de Bayes est utilisé ici pour calculer les probabilités de différents événements. Ce théorème, décrit en équation 1.2, permet de calculer des distributions de probabilité conditionnelles pour un ensemble de variables en interaction.

$$P(H|E) = \frac{(P(E|H) \times P(H))}{P(E)} \tag{1.2}$$

où H est l'hypothèse et E est l'évidence. En général, $P(A|B)$ est interprété comme "la probabilité de A étant donné B", où A est la variable dépendante et B la variable indépendante. Le théorème de Bayes détermine la probabilité de notre hypothèse H en fonction des évidences E.

Pour résumer, un réseau bayésien illustre la combinaison d'une topologie (graphe) et des probabilités conditionnelles des variables (nœuds). Ces éléments sont conjointement exploités pour examiner les interactions entre les différentes variables.

Dans certaines situations, la détermination des probabilités est aussi simple que leur attribution à l'aide de tables de distribution de probabilités conjointes. Cependant, en présence de réseaux bayésiens, ces probabilités seront adaptées par le biais d'un apprentissage au fil du recueil des données. Ce processus d'adaptation fera partie des fondements de notre projet.

1.6.2 Inférence en réseau bayésien

L'inférence consiste à calculer les probabilités d'événements inconnus dans un réseau bayésien à partir des données relatives aux événements connus (Stephenson, 2000). Concrètement, l'inférence est le processus de calcul d'une distribution de probabilité d'intérêt, par exemple $P(A|B = True)$, ou $P(A, B|C, D = True)$. L'inférence est primordiale pour déterminer les valeurs les plus probables des variables, puis en tirer des conclusions.

Les notions de prédiction (axée sur l'inférence des sorties à partir des entrées) et de diagnostic (inférence des entrées à partir des sorties) constituent deux formes connues d'inférence dont la sémantique diffère légèrement.

À titre indicatif, citons quelques exemples d'inférence en pratique :

- compte tenu d'un certain nombre de symptômes, quelles sont les maladies les plus probables ?
- quelle est la probabilité qu'un constituant tombe en panne, considérant l'état actuel du système ?
- étant donné le comportement récent de deux actions, comment se comporteront-elles ensemble au cours des cinq prochains laps de temps ?

1.6.3 Apprentissage bayésien

Les réseaux bayésiens assurent un processus d'apprentissage et compréhension des relations causales. Ce processus est particulièrement utile lors de l'exploration et de l'acquisition de nouvelles données puisque il permet la validation des hypothèses à partir des données. Le principal avantage des réseaux bayésiens dans ce contexte est la faculté d'évaluer si les répercussions d'une nouvelle particularité sont désirables ou non. Par exemple, une entreprise peut évaluer si une nouvelle campagne de marketing a eu un impact significatif sur les ventes en se servant des statistiques bayésiennes pour comparer les anciennes et les nouvelles chiffres de vente.

Les réseaux bayésiens sont particulièrement efficaces pour combiner une connaissance a priori du domaine aux données. Autrement dit, ils sont en mesure soit de se servir des données pour évaluer la validité des connaissances préalables, soit de redresser ces dernières en tirant des conclusions des données et en affinant leurs hypothèses au sujet du modèle (Heckerman, 2008).

1.7 Conclusion

Ce chapitre a présenté les concepts et les approches adoptés pour mener notre projet. Les challenges entourant des concepts de systèmes adaptatifs, d'anomalies, de gestion de clusters de conteneurs et également des réseaux bayésiens sont notamment illustrés.

CHAPITRE 2

REVUE DE LITTÉRATURE

2.1 Introduction

Ce chapitre fournit un aperçu des travaux pertinents étudiant des analyses de performance des systèmes distribués. Nous explorons les différentes approches traitant multiples problématiques à savoir introduire les outils contrôlant des infrastructures conteneurisées capables de suivre un comportement anormal d'une composante, présenter les méthodes statistiques de détection et prédiction d'anomalies et caractériser les métriques requises pour mettre en place des modèles adaptatifs de prévention et gestion de défaillance.

2.2 Analyse d'anomalies

De nombreuses études, similaires à celle de Ghaith *et al.* (2016), croisent les modalités de détection, d'identification et de diagnostic d'anomalies alors qu'ils exsitent des nuances entre les processus. Par exemple, pour déterminer si un dysfonctionnement s'est produit, une détection anticipée peut servir pour alerter (processus de détection). Toutefois, pour déceler l'effet du dysfonctionnement, nous parlons d'un processus d'identification, capable de déterminer le type d'anomalie, sa date de production et ses causes racines. À la fin, un mécanisme de récupération servira à éliminer l'effet de l'anomalie.

Diverses approches d'analyse des anomalies ont été employées pour identifier une dégradation de performances d'un système logiciel (Maiga, Ali, Bhattacharya, Sabane, Gueheneuc & Aimeur, 2012). Pour se conformer aux exigences de performance, nous devons repérer toutes fluctuations indésirables de performance en identifiant le type d'anomalie, sa sévérité, ses causes racines et ses dépendances fonctionnelles et en appliquant un mécanisme de récupération adéquat.

Ghaith *et al.* (2016), Farshchi *et al.* (2016) et Waller, Ehmke & Hasselbring (2015) analysent la performance en se basant sur des données historiques pour identifier les éventuelles dégradations.

Cependant, ces approches ne permettent pas de déceler les causes racines des problèmes constatés.

Ibidunmoye, Metsch & Elmroth (2016) détectent un comportement anormal au moyen d'un modèle de prédiction pour repérer les fluctuations de performance. Toutefois, ce travail emploie un seuil rigide pour l'ensemble des données, ne reflétant pas la consommation réelle dans un système.

En outre, il existe multiple approches (Nistor, Song, Marinov & Lu, 2013; Yan, Xu & Rountev, 2012) capables de détecter des anomalies de performance au cours du développement. Cependant, certaines se focalisent sur des anomalies spécifiques et ne permettent pas une démarche généraliste de diagnostic.

Ibidunmoye *et al.* (2016) développent modèle descriptif de diagnostic des défauts de performance capable de spécifier les informations requises pour effectuer un diagnostic automatique des dysfonctionnements, néanmoins, ce formalisme ne fournit pas un mécanisme de récupération des défauts détectés, ni découvre la dépendance entre les anomalies. De plus, l'approche se base sur des heuristiques prédéfinies pour détecter les problèmes de performance. Par conséquent, appliquer l'approche sur un domaine différent ou modifier le modèle de défaillance nécessite une mise à jour des heuristiques.

Wert (2015) classe les défauts de performance en trois catégories, à savoir : les symptômes, indicateurs visibles d'une anomalie, évidences, indicateurs internes de performance et enfin les causes racines, dont la neutralisation élimine les évidences et les symptômes d'un dysfonctionnement. Toutefois, la méthode ne considère pas la dépendance entre les anomalies et necessite une expertise humaine pour fournir les heuristiques nécessaires pour détecter les problèmes de performance.

Wang, Xu, Zhang, Gu & Zhong (2018) et Guan, Chiu & Fu (2016) analysent les corrélations entre diverses métriques pour diminuer le nombre de mesures surveillées, réduire la complexité de surveillance et identifier les métriques pertinentes pour détecter les éventuelles défaillances.

Rastogi & Hendler (2017) proposent un outil permettant de déceler la densité d'une anomalie dans un graphe de nœud. Sorkunlu, Chandola & Patra (2017) identifient des anomalies de performance du système en analysant les corrélations dans les données de consommation des ressources.

Le travail dans Düllmann (2016) fournit une approche automatique d'évaluation de performance en détectant des anomalies par le biais d'une analyse des séries chronologiques discrètes des métriques. L'approche spécifie un score de comportement anormal, déclenche une alarme une fois l'anomalie détectée, et localise la cause racine d'une telle anomalie. Cependant, l'approche ne prédit pas un futur comportement anormal et détecte l'anomalie dans un environnement dynamique infonuagique.

Alcaraz, Kaanich & Sauvanaud (2016) proposent une approche détectant un comportement anormal et sa cause racine pour une application hébergée en environnement infonuagique. Toutefois, l'approche ne traite pas la dépendance entre les anomalies et ne prévoit pas une action de récupération.

2.3 Approches statistiques de détection, identification et prédiction d'anomalies

Chalapathy, Menon & Chawla (2018) ont étudié l'utilisation d'un réseau de neurones artificiel pour détecter les anomalies. Cependant, ce type d'approche nécessite une quantité considérable de données d'entraînement pour obtenir des résultats assez précis, ce qui entraîne un long délai d'opération du modèle.

L'approche de Mariani, Monni, Pezze, Riganelli & Xin (2018) consiste à surveiller les mesures à l'aide d'algorithmes à base de graphes afin de suivre les anomalies. Le travail a ciblé la causalité entre les mesures anormales afin de saisir les ressources anormales potentielles.

Le travail de Sorkunlu *et al.* (2017) partage la même vision et cible les corrélations entre les données d'utilisation des ressources afin d'identifier avec précision la dégradation en performances du système.

Le travail de Sauvanaud, Kaâniche, Kanoun, Lazri & Silvestre (2018) vise à détecter les comportements anormaux des applications déployées sur les machines virtuelles dans une infrastructure infonuagique, en émulant des injections d'anomalies basées sur une exploitation stressante du processeur, de la mémoire, du disque et du réseau. Leur travail néglige les dépendances entre les différents nœuds.

Les travaux de Guan *et al.* (2016) mettent en œuvre une approche de prédiction probabiliste visant à caractériser la corrélation entre la performance en infonuagique et les événements d'anomalies. La détection de comportements anormaux dans l'infrastructure des nuages a été réalisée en suivant la corrélation entre différentes mesures (CPU, mémoire, disque et réseau). L'auteur met à jour les distributions de probabilités conditionnelles (CPD) de chaque mesure sur les occurrences d'anomalies, et sélectionne les mesures dont les CPD dépassent un seuil statique prédéfini. Les résultats expérimentaux ont démontré une faible efficacité de prédiction.

Tan, Nguyen, Shen, Gu, Venkatramani & Rajan (2012) exposent un algorithme d'apprentissage statistique qui combine deux modèles de chaîne de Markov avec des réseaux bayésiens augmentés par arbre. Les auteurs appliquent l'approche résultante à des mesures au niveau du système (CPU, mémoire, statistiques d'entrées/sorties du réseau) pour prédire un comportement anormal potentiel.Cependant, ils ne révèlent pas d'informations sur l'efficacité de la prédiction.

Wert (2013) présente un modèle de description des performances qui spécifie les informations correspondantes pour automatiser le diagnostic des problèmes de performances. Cependant, la proposition n'implique pas la dépendance entre les anomalies.

Le travail de Bali, Al-Osta, Ben Dahsen & Gherbi (2020), qui illustre une approche réactive basée sur des règles, présente des difficultés pour réappliquer la méthode dans un contexte différent, en raison des exigences statiques de mise à jour des règles.

Le travail de Mariani *et al.* (2018) utilise la théorie des graphes pour modéliser la corrélation entre les ressources et saisir le potentiel parmi les ressources anormales. L'auteur cible les anomalies

liées au taux d'occupation de la mémoire dans un serveur physique et à la consommation de CPU d'une machine virtuelle. Cependant, il ne cible pas l'anomalie dans les conteneurs.

À la différence des des études évoquées dans les deux sous-sections précédentes, notre proposition peut :

– détecter, analyser et remédier à une dégradation de performances des composants d'une architecture distribuée.
– déterminer les dépendances entre les anomalies en retraçant les scénarios de propagation de l'anomalie.
– appliquer un mécanisme de redressement automatisé.

2.4 Conclusion

Ce chapitre aborde les travaux connexes au sujet de notre projet de recherche en exposant multiple questions relatives à la détection, l'identification et prédiction des anomalies. Cette revue présente au début un aperçu de l'analyse des anomalies en se focalisant sur la méthodologie suivie par chaque étude. Elle explore ensuite plusieurs approches spécifiques à la détection, identification et prédiction d'anomalies.

En guise de conclusion, des travaux complémentaires sont requis pour améliorer l'analyse des comportements anormaux. Comme nous l'avons constaté dans le chapitre précédemment, des décisions d'experts sont indispensables pour spécifier la configuration à exécuter pour détecter les causes du problème, ou de disposer du code source du système pour retracer et diagnostiquer les anomalies qui peuvent entraîner des défaillances, ce qui est considéré comme impossible dans certains cas (par exemple, obtenir des données d'une tierce partie). En outre, dans une architecture distribuée, la dégradation des performances se manifeste pour différents motifs (mauvaise configuration, défaillance d'un conteneur, etc). Pour gérer les dysfonctionnements dans un tel environnement, divers facteurs affectent la précision de la résolution des anomalies (par exemple, le nombre d'anomalies décelées correctement, l'efficacité du scénario de relance

appliqué en matière de coûts et délais). Malgré les nombreuses contributions en gestion dynamique des anomalies dans les environnements distribués, une partie importante de ces propositions ne reflète pas suffisamment ces facteurs. Pour pallier les implications d'un tel environnement, il est primordial de se doter d'une solution de gestion automatique des anomalies de performance répondant aux principes suivants :

- **adaptation** : la capacité de la proposition à s'appliquer à différents systèmes, à s'adapter à plusieurs variantes d'anomalies, et à considérer la dépendance entre les éléments d'un système et leurs charges.
- **prédiction** : une proposition doit anticiper un éventuel comportement anormal afin de réduire les coûts de ré-allocation des ressources et de procéder à une action de redressement appropriée.

Dans ce contexte, notre approche a pour contributions :

- traiter les divers profils d'anomalies.
- considérer les interdépendances entre les composantes d'une architecture distribuée.
- correspondre un dysfonctionnement observé à ses causes racines dans une structure hiérarchique.
- prédire un éventuel comportement anormal.

CHAPITRE 3

NOTRE APPROCHE BAYÉSIENNE ADAPTATIVE

3.1 Introduction

Dans ce chapitre nous décrivons l'architecture générale de notre approche probabiliste adaptative. Cette approche a pour cible la caractérisation de la dégradation des performances des systèmes infonuagiques en anticipant les anomalies et en analysant leurs dépendances causales. Le chapitre débute par une brève explication des modules de notre architecture adaptative générale . Ensuite, il détaille chaque module de la structure.

3.2 Modularité de l'architeture

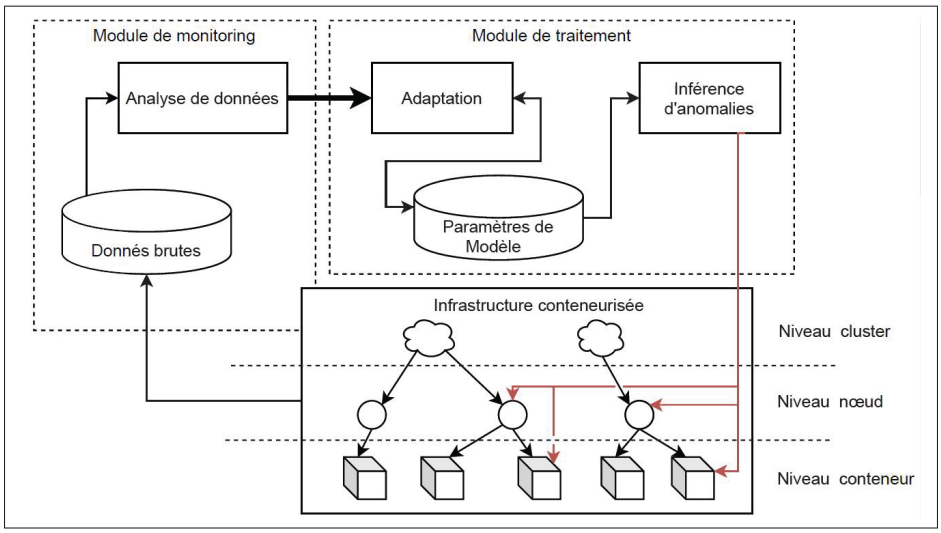


Figure 3.1 Architecture générale

L'architecture proposée, comme le montre la figure 3.1, se décompose en module de monitoring, englobant les mécanismes qui capturent les mesures de la consommation des ressources de chaque constituant de l'infrastructure. Les métriques recueillies reflètent des informations brutes sur le système (c'est-à-dire les clusters, les nœuds et les conteneurs). Pendant cette phase, nous exploitons les données historiques caractérisant un fonctionnement nominal de l'infrastructure

pour fournir les paramètres initiaux (distributions de probabilités conditionnelles) requis pour concevoir notre modèle adaptatif. Cependant, les captures qui suivront au cours du temps seront analysées, d'une part, pour détecter un comportement anormal sur une ressource considérée en fonction des seuils dynamiques, et d'autre part, pour fournir au module de traitement une mise à jour réelle sur l'état actuel afin d'adapter le modèle probabiliste.

Par ailleurs le module d'adaptation administre les mécanismes qui analysent et identifient les comportements anormaux. Ces mécanismes mettent à jour et adaptent le modèle probabiliste et prédisent un futur comportement anormal potentiel. En se basant sur les captures de consommation des ressources fournies par le module de monitoring, ce module adapte distributions de probabilités conditionnelles du modèle adaptatif ensuite évalue les anomalies potentielles. Si une anomalie se présente, un processus d'inférence aura lieu pour analyser sa dépendance causale et prédire ses pistes de propagation possibles afin de préparer un scénario de récupération proactive. Ce scénario consiste à modifier les allocations de ressources au niveau des constituants qui seront sujets à des anomalies futures selon les résultats du processus d'inférence. Cette action corrective sera ensuite évaluée en suivant l'état du système à travers le module de monitoring.

Nous alignons notre architecture par rapport à la chaîne de contrôle automatique MAPE-K. Cette chaîne est vue comme un gestionnaire autonome comportant plusieurs modules interconnectés qui coopèrent pour atteindre une autonomie d'un système

3.3 Infrastructure conteneurisée

Un cluster est constituée d'un ensemble de nœuds. Chaque nœud possède un ou plusieurs conteneurs liés aux applications déployées.

L'infrastructure considérée dans le cadre de notre projet suit trois niveaux :

– niveau des clusters, un cluster gère un ou plusieurs nœuds. Un nœud maître supervise et attribue le travail aux nœuds.

– niveau des nœuds, caractérise un dispositif virtuel qui se compose d'un ou de plusieurs conteneurs et possède une capacité prédéfinie représentant ses ressources.
– niveau des conteneurs, caractérise les instances d'une application ou d'un service qui fonctionnent dans un environnement informatique.

Nous assumons qu'un ou plusieurs nœuds peuvent appartenir à un cluster mais qu'ils sont susceptibles de se déplacer vers un autre cluster. Pour permettre la gestion des éléments du système, nous recueillons, dans le cadre de ce travail, les métriques principalement liées à l'utilisation des ressources telles que l'utilisation du processeur ou de la mémoire.

3.4 Module de monitoring

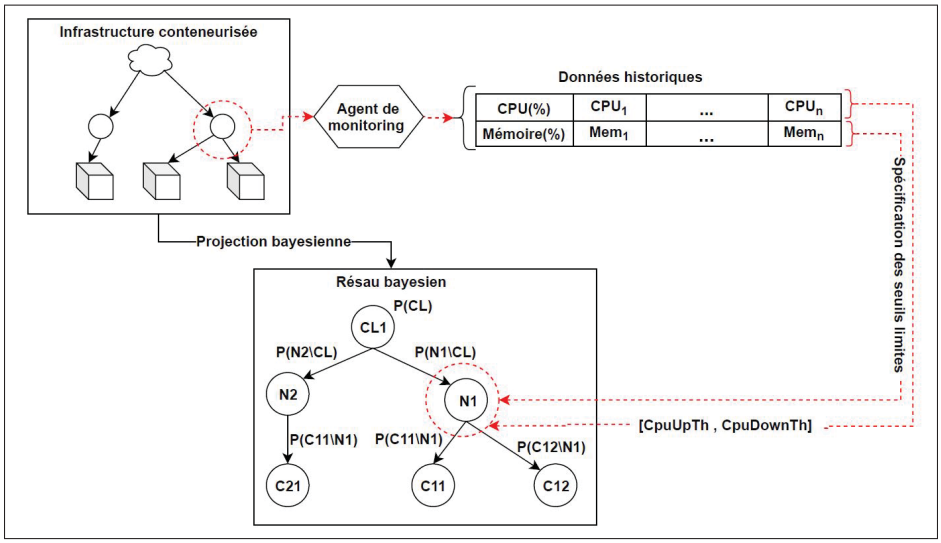


Figure 3.2 Module de monitoring

Ce module, décrit en figure 3.2, réunit les mécanismes qui collectent les mesures des ressources gérées. Les métriques capturées se divisent en données historiques et données en flux continu. Les données historiques évaluent un fonctionnement nominal de notre système infonuagique. Ils servent en premier lieu à estimer les paramètres initiaux (distributions de probabilités conditionnelles) requises pour initialiser le modèle bayésien. Cette estimation est implémentée en se basant sur les fréquences relatives auxquelles les états des variables se sont produites.

Par exemple si nous avons observé sur 100 essais qu'un nœud présente une anomalie de fonctionnement, nous pouvons donc supposer que la probabilité de défaillance de ce nœud est égale à 0.01. En second lieu, ces données historiques serviront pour calculer et mettre à jour les seuils limites de consommation des ressources qui alertent la présence d'un comportement anormal sur un nœud de l'architecture. Cette dernière est sujette à des modifications en matière de nombre de nœuds et conteneurs

L'algorithme 3.1 résume le processus des spécification des seuils limites de consommation des ressources.

Algorithme 3.1 Spécification des seuils limites de consommation

Input : BayesNet$[Node_1, .., Node_i]$,
Resources$[CPU, Memory, Network]$,
NominalData $[< Node_i, [CPU[], Mem[], Net[]] >$

Output : MaxThreshList$[< Node_i, .., [CpuUpTh, MemUpTh, NetUpTh] >]$,
minThreshList$[< Node_i, .., [CpuUpTh, MemUpTh, NetUpTh] >]$

$SafeMargin \leftarrow 5$;
$HighCritical \leftarrow 98$;
$LowCritical \leftarrow 2$;
for *each* $Node_i \in BayesNet[]$ **do**
  for *each* $Res_j \in Resources[]$ **do**
    $MedianCpuN_i \leftarrow ComputeMedian(Node_i, NominalData < Node_i, Res_j[] >)$;
    $MAD \leftarrow ComputeMAD(MedianCpuN_i, NominalData < Node_i, Res_j[] >) + MedianRes_jN_i$;
    $MaxThreshList[Node_i, Res_j] \leftarrow min((MAD + SafeMargin), HighCritical)$;
    $minThreshList[Node_i, Res_j] \leftarrow Max((MAD - SafeMargin), LowCritical)$;
  end for
end for

Nous commençons tout d'abord par exprimer les inputs de l'algorithme, notamment le réseau bayésien, les ressources à surveiller et les données historiques retraçant un fonctionnement

nominal d'une architecture conteneurisée. Nous nous attendons à retourner deux listes comportant une correspondance entre les nœuds et leurs seuils limites supérieurs et inférieurs de consommation de chaque ressource surveillée à savoir le CPU et la mémoire. Ensuite nous spécifions les paramètres à employer en algorithmes, notamment les pourcentages critiques de consommation des ressources ainsi que la marge de manoeuvre à considérer en calcul des seuils limites.

Le processus débute par caractériser pour chaque couple <nœud, ressource> la tendance globale des données historiques de consommation de la ressource par le nœud en question. Cette caractérisation se traduit par une mesure de médiane des mesures de consommation nominales de chaque composant de l'architecture pour chaque ressource surveillée. Nous qualifions également la dispersion de ces données par le biais d'un écart absolu médian décrite en équation 3.1.

$$MAD = \frac{1}{n} \sum_{i=1}^{n} |x_i - \widetilde{x}| \tag{3.1}$$

où, x_i désigne une valeur observée et $\widetilde{x}$ correspond à la médiane d'un échantillon de données.

Ces calculs nous permettent de définir un seuil supérieur (respectivement inférieur) comme la somme (ou soustraction) de la valeur médiane et de l'écart absolu moyen, moyennant une marge de manoeuvre de 5%. Nous plafonnons notamment les seuils par des limites critiques maximales et minimales (98% et 2%).

En second lieu, le module de monitoring incorpore une composante d'analyse du flux continu des données permettant de fournir au module d'adaptation une capture des états actuels des nœuds du réseau bayésiens. Cette analyse permet d'alerter tout franchissement des seuils limites de consommations de ressources et ainsi détecter les nœuds présentant des anomalies ce qui permettra au module de traitement à suivre et prédire la propagation de ces anomalies

3.5 Module de traitement

Le module de traitement repose sur trois grandes phases :

– la projection de l'infrastructure conteneurisée dans un formalisme de réseau bayésien. La topologie du réseau est sujette à des changements en augmentant ou diminuant le nombre de composants, ou en activant ou désactivant certains composants.
– adaptation des paramètres du réseau bayésien (distributions de probabilité conditionnelles). Ces paramètres sont mis à jour en se basant sur les informations du processus d'analyse des flux de données provenant du module de monitoring.
– inférer les anomalies potentielles, consiste à calculer la distribution de probabilité pour chaque nœud du réseau afin de prédire et d'identifier les éventuelles anomalies sur les nœuds désignés. La prédiction est effectuée en évaluant les scénarios de propagation des anomalies et en retraçant leurs causes et effets. Ceci permettra d'anticiper et d'appliquer des scénarios proactifs de prévention des anomalies potentielles prédites.

3.5.1 Projection bayésienne d'une infrastructure conteneurisée

Comme nous l'avons décrit en chapitre 1, les réseaux bayésiens représentent une approche graphique de modélisation, qui se base sur un principe probabiliste. Dans un réseau, les nœuds sont destinés à représenter des variables, et les liens servent à indiquer une éventuelle influence d'un nœud par rapport à un autre. Cela permet de visualiser clairement la relation entre les variables.

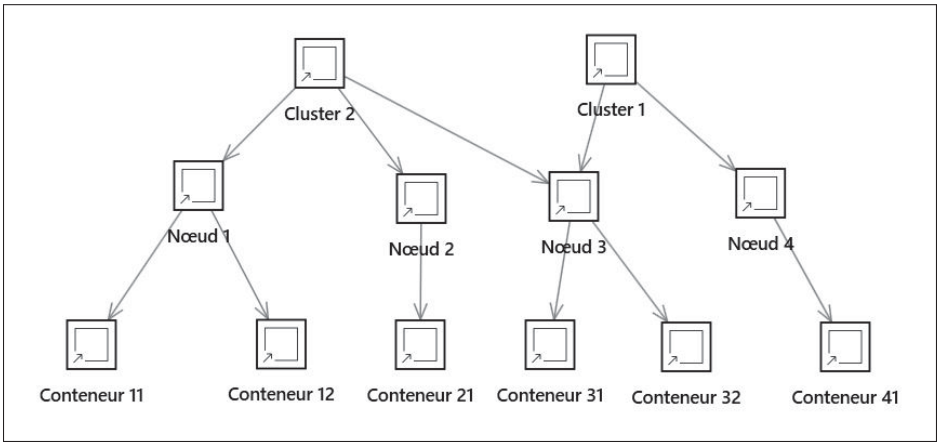
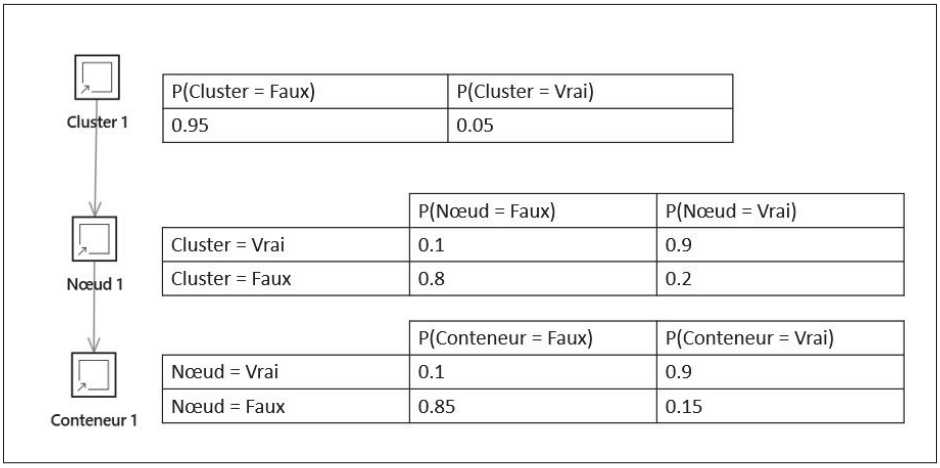


Figure 3.3 Infrastructure conteneurisée .

Nous avons adopté le formalise bayésien pour modéliser notre architecture conteneurisée : comme le révèle la figure 3.3, chaque cluster, nœud ou conteneur de l'architecture est considéré comme une variable (nœud) dans une topologie bayésienne.



P(Cluster = Faux)	P(Cluster = Vrai)
0.95	0.05

	P(Nœud = Faux)	P(Nœud = Vrai)
Cluster = Vrai	0.1	0.9
Cluster = Faux	0.8	0.2

	P(Conteneur = Faux)	P(Conteneur = Vrai)
Nœud = Vrai	0.1	0.9
Nœud = Faux	0.85	0.15

Figure 3.4 Exemple d'une distribution de probabilité dans un réseau bayésien

Comme le décrit la figure 3.4, chaque nœud d'un réseau bayésien nécessite la spécification d'une distribution de probabilité (conditionnelle à ses parents). Ce processus est appelé apprentissage des paramètres et peut-être effectué à l'aide des données historiques recueillies pendant l'étape de monitoring.

À ce stade, nous considérons notre infrastructure conteneurisée comme une structure complètement bayésienne. La suite consiste à résoudre (en se basant sur le théorème de Bayes décrit en équation 3.2), une équation de probabilité conjointe (décrite en équation 3.3) pour inférer la probabilité d'anomalie de chaque composant du système.

$$P(Node_i|NodeB) = \frac{P(NodeB|Node_i)P(Node_i)}{\sum_{j=1}^{n} P(NodeB|Node_j)P(Node_j)} \tag{3.2}$$

$$P(Node_i, ..., Node_n) = \prod P(Node_i|Parents(Node_i)) \tag{3.3}$$

3.5.2 Notre approche bayésienne adaptative

La mise en service d'une approche adaptative efficace pour ajuster les paramètres du réseau constitue le principal objectif de notre travail. En effet, nous proposons un processus d'adaptation, permettant une mise à jour incrémentale des distributions dans notre réseau bayésien, qui prépare le champ pour inférer la propagation des anomalies potentielles. La figure 3.5 illustre la logique de notre contribution.

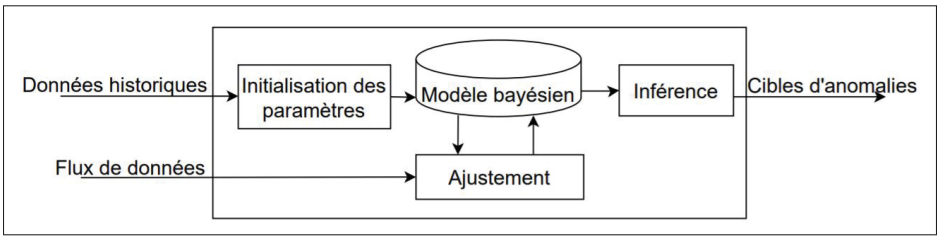


Figure 3.5 Résumé de l'approche adaptative .

Notre processus d'adaptation ou apprentissage bayésien requiert la définition d'un **taux de confiance** (TC_i) pour chaque combinaison parentale d'un nœud (N_i, $Parent(N_i)$). Par exemple, un taux de 1000 reflète une forte confiance à l'égard de la probabilité associée. , alors qu'un taux de 1 reflète une faible confiance.

En plus du taux de confiance TC_i, un **taux d'atténuation** (TA_i) variant entre [0, 1] est optionnellement spécifié pour chaque combinaison nœud/parent, afin de mettre l'accent sur l'importance des informations actuelles comparativement par rapport aux précédentes. En d'autres termes, l'importance des informations antérieures se dégrade progressivement. Une valeur TA_i de 1 signifie qu'il n'y a pas d'atténuation, tandis que des taux qui sont inférieurs à 1 indiquent que l'atténuation est appliquée.

Ayant tous les paramètres requis, nous exprimons notre équation d'adaptation sous une forme récursive décrite ci-dessous. Elle réfère à l'évolution temporelle de n à n+1 de la probabilité d'observation d'une anomalie sur un nœud désigné.

$$P_{n+1}(N_i) = P_n(N_i) * TA(N_i) \pm \frac{P_0(N_i) * TC_0(N_i)}{TC_n(N_i) + TC_n(N_i)^2} \tag{3.4}$$

Les paramètres du réseau bayésien s'adaptent aux flux entrant de données reçues de la phase de monitoring. Prenons l'exemple présent en figure 3.6, un agent de monitoring déployé pour chaque élément de notre infrastructure capture un flux continu de donnés des consommations des ressources. Chaque capture est comparée par rapport à son seuil limite correspondant afin de spécifier le taux d'ajustement ou l'adaptation à apporter à la probabilité de défaillance de l'élément en question.

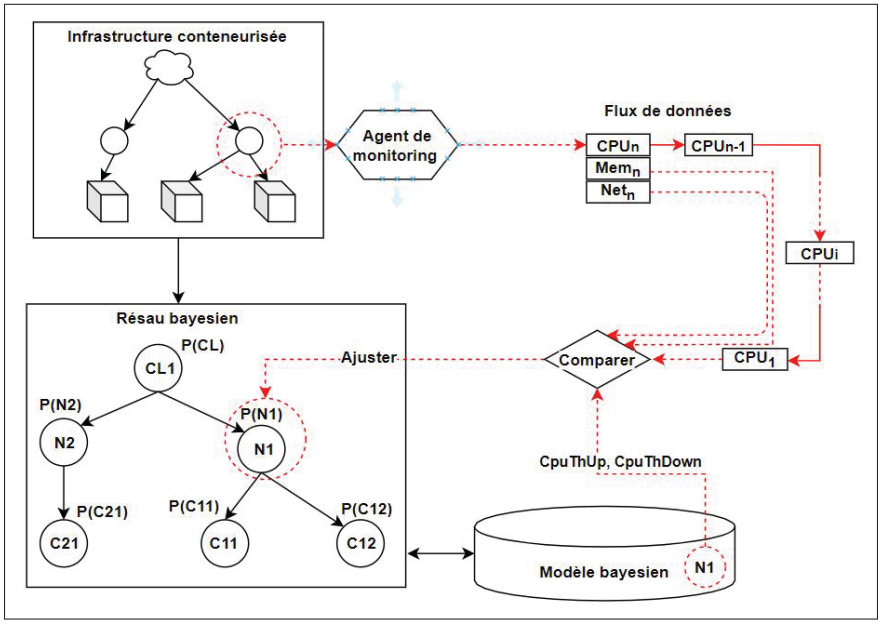


Figure 3.6 Processus d'adaptation .

Ce processus d'adaptation intervient en premier lieu en phase d'apprentissage des paramètres initiaux du modèle bayésien. En effet nous décrivons ce volet en algorithme 3.2.

Nous commençons tout d'abord par exprimer les inputs de l'algorithme, notamment le réseau bayésien, les données historiques retraçant un fonctionnement nominal d'une architecture conteneurisée et enfin la liste de correspondances entre les nœuds de l'architecture et leurs seuils

limites. Nous nous attendons à retourner une liste illustrant les distributions des probabilités conditionnelles initiales de chaque nœud du réseau.

Algorithme 3.2 Apprentissage des paramètres

Input : $BayesNet[Node_1..Node_i]$,
NominalDataPerRes $[< Node_i, Res[] >]$,
ThreshVals$[< Node_i[ResUpTh, ResDownTh] >]$

Output : NodesCPDs[<Node, [CPD,TC]>]

1 $TA \leftarrow 0.95$;
2 **for** ***each*** $Node_i \in \mathcal{B}ayesNet[]$ **do**
3 $\quad NodesCPDs[Node_i].CPD] \leftarrow 0.05$;
4 $\quad NodesCPDs[Node_i].TC] \leftarrow 1$;
5 **end for**
6 **for** ***each*** $ResValue \in \mathcal{N}ominalDataPerRes.Res[]$ **do**
7 $\quad NodesCPDs[Node_i] \leftarrow$ **Update**$(NodesCPDs[Node_i])$;
8 **end for**

Nous assumons un taux d'atténuation de 0.95 applicable pour tous les nœuds du réseau.

Pour chaque nœud du réseau, nous correspondons initialement une probabilité de défaillance 0.05 assumant une hypothèse qu'un élément de l'architecture est sujet à 5 dysfonctionnements au cours des 100 traitements subis. Nous qualifions une faible confiance vis-à-vis de cette probabilité par un taux de confiance égal à 1. Ensuite pour chaque capture provenant des données historiques, nous ajustons la distribution de probabilité du nœud en question et nous incrémentons son taux de confiance.

Nous projetons également cet ajustement ou adaptation en second lieu pour toute capture de flux continu de données reçues de la phase de monitoring. L'algorithme 3.3 illustre cette logique d'adaptation.

En effet, nous débutons par exprimer les inputs de l'algrithme, à savoir un couple clé valeur reliant un nœud à ses paramètres bayésiens (son CPD et son taux de confiance), une capture récente de la consommation d'une ressource et enfin les seuils limites alertant un épuisement ou

une sous-utilisation des ressources disponibles. Nous nous attendons à adapter les paramètres bayésiens du nœud étudié. Nous qualifions cette mise à jour comme une forme récursive d'évolution temporelle de n à n+1 de la probabilité d'observation d'une anomalie sur un nœud désigné et de l'augmentation de son taux de confiance associé. En effet pour passer de n à n+1 l'équation d'adaptation 3.4 comporte un facteur fixe qui représente une multiplication par un taux d'atténuation reflètant l'atténuation à appliquer sur l'instance de probabilité P_n par rapport à $P_(n+1)$. Selon une règle de décision impliquant le franchissement ou non des seuils limites par une capture de consommation des ressources d'un nœud désigné, nous additionnons ou soustrayons un facteur de pénalisation proportionnelle à la probabilité initiale et au taux de confiance du nœud en question. Par exemple si nous recevons une capture indiquant que la consommation actuelle en matière de CPU d'un nœud désigné dépasse le seuil limite supérieure, dans ce cas nous sommes face à une anomalie d'épuisement de la ressource CPU, donc nous attendons à conceptualiser cette anomalie en ajustant la probabilité de défaillance de ce dernier en l'incrémentant par le facteur de pénalisation.

Algorithme 3.3 Adaptation des paramètres

Input : $< N_i, [CPD, TC] >, TA(N_i), ResVal$, ThreshUp, ThreshDown

Output : $< N_i, [CPD, TC] >$

$P_n(N_i) \leftarrow Node_i.CPD$;
$P_0(N_i) \leftarrow 0.1$;
$TC_0(N_i) \leftarrow 1$;
$TC_n(N_i) \leftarrow TC_0(N_i) + 1$;
if $(ResVal > ThreshUp \;||\; ResVal < ThreshDown)$ **then**
    $Node_i.CPD \leftarrow P_n(N_i) * TA(N_i) + \frac{P_0(N_i) * TC_0(N_i)}{TC_n(N_i) + TC_n(N_i)^2}$;
    $TC_n(N_i) + +$;
end if
if $(ResVal < ThreshUp \&\& ResVal > ThreshDown)$ **then**
    $Node_i.CPD \leftarrow P_n(N_i) * TA(N_i) - \frac{P_0(N_i) * TC_0(N_i)}{TC_n(N_i) + TC_n(N_i)^2}$;
end if

3.5.3 Inférence

Inférer les anomalies potentielles revient à évaluer les scénarios de propagation des anomalies détectés sur des nœuds du réseau bayésien, à savoir retracer les effets potentiels de ces anomalies sur les autres nœuds. Pour ce faire, nous prenons comme évidence la présence d'une anomalie détectée sur un nœud désigné, c'est-à-dire nous qualifierons la valeur de sa probabilité à $P(N_i) = 1$ et nous suivrons l'effet de cette qualification les probabilités de défaillance des autres nœuds du réseau. Ce fait permet de cibler et localiser les futures anomalies possibles et ainsi anticiper un plan de prévention proactif.

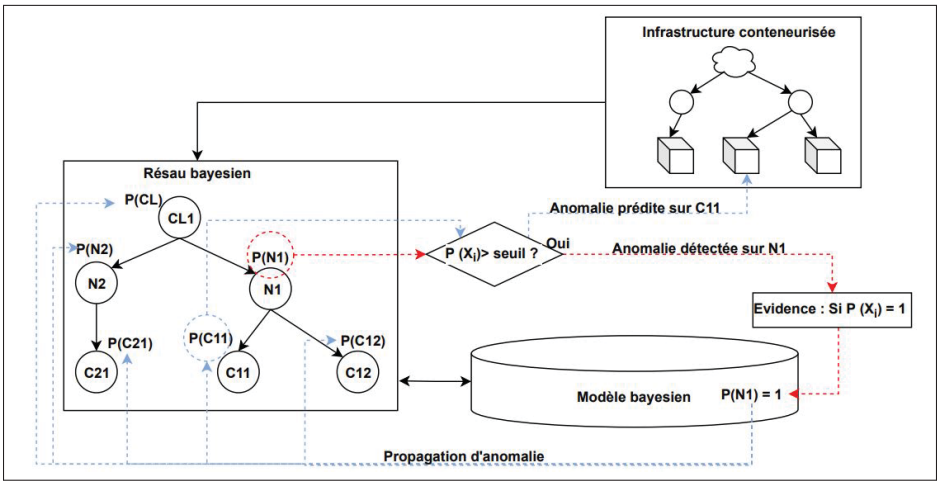


Figure 3.7 Processus d'inférence .

Nous traitons l'exemple indicatif présent en figure 3.7. Nous remarquons que la probabilité de défaillance du nœud N1 dépasse un seuil limite donc nous pouvons qualifier un état d'anomalie détectée sur ce nœud. À ce stade, nous projetons sur le plan bayésien que si anomalie survient sur N1 ($P(N1) = 1$) que seront les effets sur les autres nœuds de l'architecture ? Nous répondons à cette question en inférant les valeurs de probabilité des nœuds du réseau en prenant comme évidence la présence d'une anomalie sur le nœud 1, autrement dit nous suivons la propagation de l'anomalie vécue sur le nœud N1. Dans notre exemple, nous évaluons que le nœud C11 présentera des signes d'anomalie vue sa forte dépendance causale traduite par le modèle bayésien adaptatif. Dans ce cas nous pouvons anticiper un scénario de récupération proactive ciblant

un allégement des ressources au niveau C11 afin de contourner son potentiel état d'anomalie prédite.

Nous développons le scénario correctif proactif à adopter en figure 3.8. Il continue la description de l'exemple précédent. Notre point de départ est la prédiction d'un excès de consommation potentiel au niveau du nœud C11 du réseau bayésien. Nous ciblons donc comme action préventive, une augmentation des ressources disponibles pour C11. Cette action a pour effet la compensation du futur besoin potentiel de CPU du nœud C11 et donc le non franchissement des seuils limites de consommation CPU ce qui implique directement la diminution de la valeur de la probabilité de présence d'anomalies au niveau C11. Ce fait implique indirectement la correction réactive de l'anomalie détectée initialement sur N1 vue la forte dépendance causale entre C11 et N1.

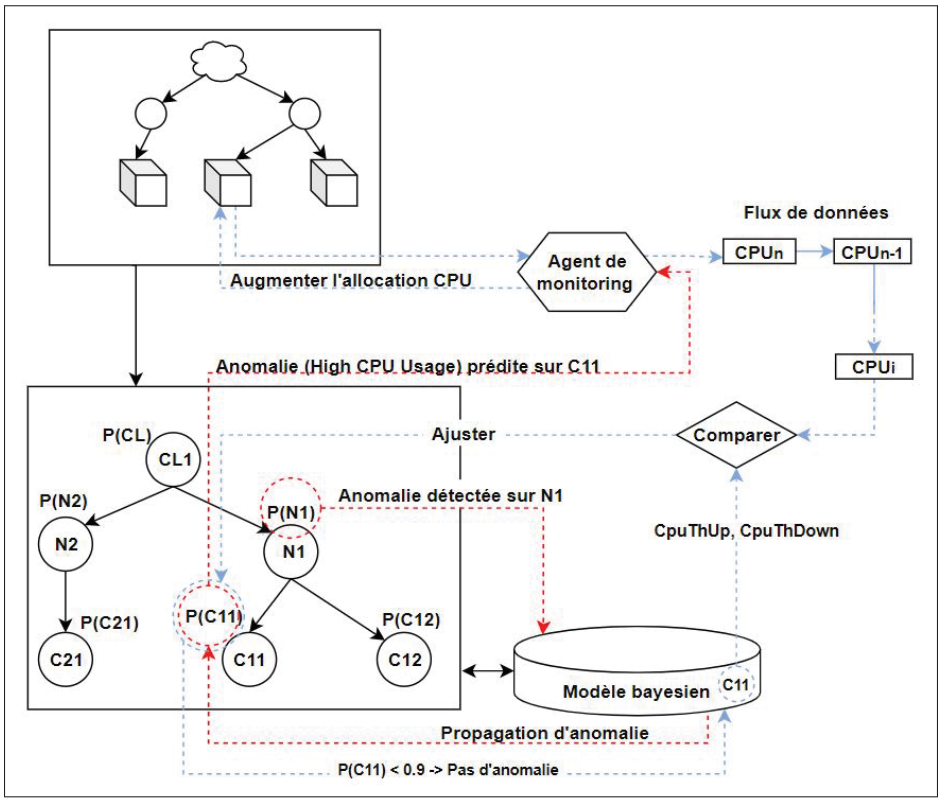


Figure 3.8 Scénario de récupération d'anomalie prédite .

3.6 Conculsion

Ce chapitre donne un aperçu des principaux volets de l'approche adaptative proposée. Cette dernière est constituée du module de monitoring, responsable de collecter les données sur les composants de l'architecture et leurs ressource dans le but de détecter toute déviation anormale. Cependant, le module de traitement est responsable de traiter les informations collectées par l'étape de monitoring pour remonter à la cause racine du comportement anormal. Si un comportement anormal se présente, un processus d'inférence aura lieu pour analyser ses dépendances causales et prédire ses pistes de propagation afin de préparer un scénario de récupération proactive.

Nous alignons notre approche par rapport à la chaîne de contrôle automatique MAPE-K. Ce formalise comporte plusieurs modules interconnectés qui coopèrent pour atteindre une autonomie globale d'un système donné. Pour permettre la gestion des éléments de ce système, nous recueillons, dans le cadre de notre proposition, les métriques principalement liées à l'utilisation des ressources telles que l'utilisation du processeur ou de la mémoire.

CHAPITRE 4

EVALUATION

4.1 Introduction

Pour évaluer l'architecture, multiples scénarios d'anomalie provoquant un stress au niveau du CPU et de la mémoire sont injectés à différents niveaux du système (nœud, conteneur). Des applications Docker sont utilisées pour fournir synthétiquement une charge de travail modérée ou intense.

L'approche adaptative proposée est évaluée en utilisant diverses métriques sur la base de :

– précision : le nombre d'anomalies détectées et prédites.
– délais : le temps nécessaire à l'approche pour détecter, et prévenir une anomalie.

Ce chapitre est structuré comme suit : la section 4.2 expose la configuration expérimentale. Les sections 4.3 et 4.4 décrivent le plan d'expérimentation, qui implémente des scénarios d'anomalie injectés dans les différents composants du système. La section 4.5 détaille les métriques appliquées pour évaluer l'architecture proposée. La section 4.6 aborde l'évaluation de l'approche proposée, en partant de la détection d'une anomalie jusqu'à sa récupération.

4.2 Configuration expérimentale

Cette section explore :

– l'utilitaire TPC-W [3] qui spécifie une charge de travail qui simule des clients naviguant et achetant des produits sur un site web. Cette charge repose sur une interaction entre un serveur Web, un serveur client et un serveur de base de données, pour fournir une solution e-commerce complète.
– la plateforme d'expérimentation

[3] http ://www.tpc.org/tpcw/

– la collecte des données.

4.2.1 Utilitaire TPC-W

TPC-W simule une librairie en ligne qui supporte des opérations d'e-commerce, et il se compose de 3 piliers : une application client, un serveur web et une base de données. Le modèle d'affaire est vu comme un détaillant qui vend des produits et des services en ligne. Le site permet à ses clients de parcourir les produits (par exemple, les meilleures ventes ou les nouveaux produits), de recueillir des informations, de consulter les détails, de placer une commande ou de vérifier son état. Les clients doivent s'inscrire sur le portail pour pouvoir acheter.

TPC-W spécifie que le portail entretient une base de données contenant des informations sur les clients, les articles du catalogue, les commandes et les transactions effectuées.

TPC-W a deux interactions web :
– interactions de navigation : parcourir, sélectionner ou rechercher des produits.
– interactions de commande : placer, confirmer, afficher ou suivre une commande.

4.2.2 Plateforme d'expérimentation

L'environnement d'expérimentation se compose d'un cluster contenant des nœuds, un manager et des workers (voir figure 4.1), déployés sous forme d'un Docker swarm. Chaque nœud est doté d'un certain nombre de conteneurs.

L'infrastructure est définie et projetée comme un réseau bayésien en utilisant la bibliothèque PGMPY [4] de python. Nous avons utilisé package python pour coder l'approche adaptative d'apprentissage bayésien.

L'utilitaire TPC-W est déployé sur des conteneurs, dont le nœud 1 héberge le serveur web, le nœud 2 héberge l'application client et le nœud 3 héberge la base de données.

[4] https ://pgmpy.org/

L'expérimentation est mené sur un PC équipé du processeur Intel i5 1.8GHz, 16 Gbits de RAM, Ubuntu 20.04.1 LTS version x64, et 500 Gbits de stockage.

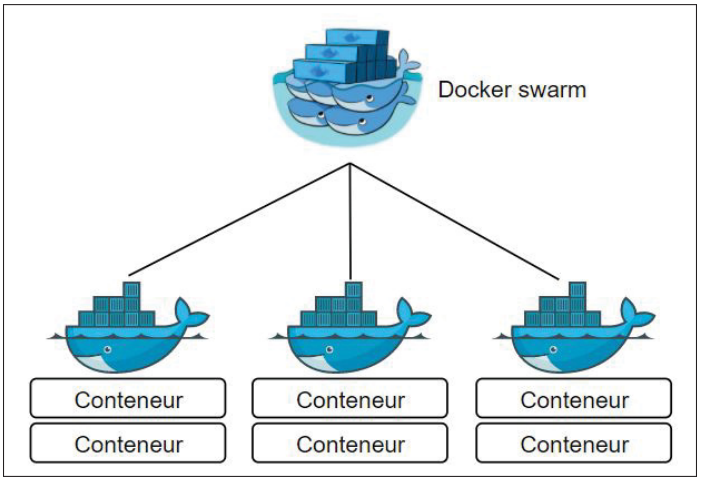


Figure 4.1 Concept de Docker swarm .

Pour détecter et identifier des anomalies affectant les conteneurs et nœuds de l'architecture, l'initialisation, l'adaptation de notre approche bayésienne adapdative a duré 180 minutes pour produire un ensemble des résultats d'évaluation. La performance de l'architecture proposée est analysée suivant deux méthodes : Premièrement, appliquer divers scénarios de tests pour mesurer la prestation du système. Deuxièmement, évaluer l'efficacité de cette contribution à détecter, prédire et récupérer des comportements anormaux.

4.2.3 Collecte des données de performance

Tableau 4.1 Liste des métriques surveillées

Métrique	Description
CPU Usage	Utilisation du CPU sur tous les noyaux
CPU Request	Besoin du CPU en millicores
CPU Limit	Limite du CPU en millicores
Memory Usage	Utilisation de mémoire
Memory Request	Besoin de mémoire en octets
Memory Limit	Limite de mémoire en octets

Étant donnée le grand nombre de métriques à surveiller sur différents niveaux de l'architecture, l'identification des principales métriques est nécessaire pour décrire convenablement la performance globale du système étudié. Le tableau 4.1 illustre les mesures collectées aux différents niveaux du système.

InfluxData [5], une pile logicielle complète de monitoring est utilisée et configurée pour observer les performances d'exécution des composants et de leurs ressources. Ce framework comprend trois modules principaux :

– Telegraf [6] , un agent serveur piloté par des plugins pour la collecte et la communication des métriques.
– InfluxDB [7], une base de données de séries temporelles pour les mesures, les événements et l'analyse en temps réel
– Chronograf [8], un outil de visualisation en temps réel pour construire des graphiques à partir de données.

L'expérimentation porte sur la consommation du CPU et de la mémoire. Ces mesures sont recueillies à partir du serveur web. Telegraf sert à récupérer le flux continu de données de performance des conteneurs en cours d'exécution. Ces données déterminent le taux d'utilisation du CPU, obtiennent des statistiques d'utilisation de la mémoire.

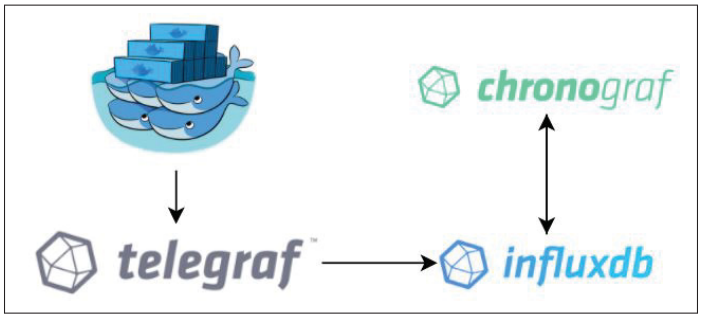


Figure 4.2 Architecture logicielle de monitoring .

[5] https ://www.influxdata.com/

[6] https ://www.influxdata.com/time-series-platform/telegraf/

[7] https ://www.influxdata.com/products/influxdb/

[8] https ://www.influxdata.com/time-series-platform/chronograf/

Comme le montre la figure 4.2, les données recueillies par Telegraf sont stockées dans une base de données InfluxDB. Cette base est interrogée pour obtenir l'état de consommation des ressources d'un élément spécifique de l'infrastructure.

Les métriques du tableau 4.1 sont observées avant, en cours et après l'application de notre approche bayésienne adaptative. Pour chaque métrique, des seuils dynamiques sont spécifiés en suivant l'algorithme 3.1 du chapitre précédent pour indiquer la présence d'un comportement anormal d'une ressource d'un composant du système.

4.3 Scénarios d'anomalies

La mise en service de notre approche bayésienne adaptative requiert un flux continu des données pour initialiser et adapter les paramètres du modèle. Cependant, la disponibilité des données représentatives qui rapproche notre modélisation au contexte réel est limitée en raison de la précarité des occurrences de dysfonctionnement et la complexité de les retracer manuellement. Dans cette section, des situations d'épuisement de ressources sont simulées pour créer un stress sur des nœuds ou conteneurs spécifiques de l'architecture. En effet, nous injectons divers profils d'anomalies au niveau des conteneurs et des nœuds en vue de constituer un contexte réaliste pour notre expérimentation.

Nous formulons un script, tel que décrit en algorithmes 4.1 et 4.2, pour injecter des scénarios d'anomalies sur le système sous test (SUT - System Under Test).

Algorithme 4.1 Algorithme d'injection d'anomalie au niveau des conteneurs

Input : N :liste des nœuds,
C : Liste des conteneurs,
A : Liste d'anomalies,
A_d : Durée d'injection,
P : Pause

Output : $Stauration\_CPU$, Fuite_Mémoire at C_j

```
for each N_i ∈ 𝒩 do
    for each C_j ∈ C ) do
        for each A_k ∈ 𝒜 do
            Injecter_Anomalie(A_k, C_j, P)
            Pause(P)
        end for
    end for
end for
```

Algorithme 4.2 Algorithme d'injection d'anomalie au niveau des nœuds

Input : N :liste des nœuds,
C : Liste des conteneurs,
A : Liste d'anomalies,
A_d : Durée d'injection,
P : Pause

Output : $Stauration\_CPU$, Fuite_Mémoire at N_i

```
for each N_i ∈ 𝒩 do
    for each A_k ∈ 𝒜 do
        Injecter_Anomalie(A_k, N_j, P)
        Pause(P)
    end for
end for
```

Les anomalies sont simulées à partir de plusieurs outils de stress pour démontrer le comportement de la solution proposée sous diverses charges. L'outil docker stress [9] est utilisé pour contraindre

[9] https ://hub.docker.com/r/progrium/stress/

les ressources CPU et mémoire en appliquant différentes configurations sur des nœuds ou conteneurs cibles de l'architecture.

Nous appliquons un profil d'anomalie à la fois. L'injection d'anomalie dure au moins 5 minutes (300 secondes) afin de garantir la collecte d'une quantité suffisante de données pour valider le modèle d'apprentissage bayésien adapté. Le processus de stress est répété plusieurs fois pour obtenir les valeurs moyennes des propriétés significatives et pour vérifier les résultats obtenus

Le tableau 4.2 illustre les anomalies injectées.

Tableau 4.2 Liste des anomalies injectées

Anomalie	**Description**	**Procédure d'injection**
Saturation CPU	Consommer toutes les unités CPU	Employer des boucles infinies qui épuisent les unités CPU
Fuite de mémoire	Épuiser la mémoire d'un composant	Submerger l'application conteneurisée pour solliciter toute sa capacité mémoire

Les paragraphes qui suivent abordent les commandes utilisées pour stresser les ressources des conteneurs et des nœuds (CPU, mémoire).

4.3.1 Saturation CPU

Une surcharge de la ressource CPU conduit à un comportement anormal du constituant (par exemple, un élément qui est très sollicité.) Ce type d'anomalie est dû à une intensification imprévue de la charge de travail, à la maintenance du système ou à des limitations de ressources.

Nous nous servons de l'image Docker stress pour provoquer un épuisement des ressources CPU et mémoire. Le tableau 4.3 illustre le fait qu'un conteneur est amené à épuiser des CPU de sa VM en utilisant la commande '$--cpu$' qui stresse un '$--cpu1$' ou plusieurs CPU '$--cpu3$', avec '$--timeout\ 60s$' pour spécifier la durée du stress. L'argument '$--vmbytes$' alloue un nombre d'octets de la mémoire virtuelle d'un VM. Sa valeur par défaut correspond à 256 MB.

Tableau 4.3 Commande Docker stress

docker run $--rm$ $-it$ progrium/stress $--cpu 2$ $--io 1$ $--vm 2$ $--vm-bytes 128M$ $--timeout$ $60s$

Par exemple, la commande '$docker run --rm --name\ cpuStress\ -it\ progrium/stress\ --cpu 2 --io 2 --vm 2 --vm-bytes 256MB --timeout 60s$' indique qu'il faut charger l'image progrium/stress, créer un conteneur portant le nom cpuStress, exécuter deux VM, limiter la mémoire du conteneur à 256 Mo et enclencher deux CPU simultanément pour évaluer la situation durant 60 secondes. Nous paramétrons la commande du tableau 4.3 sous multiples configurations pour chaque conteneur ou nœud à stresser afin de créer diverses scénarios de pression.

4.3.2 Fuite de mémoire

Dans ce scénario, nous simulons une fuite de mémoire. Ceci est fait par une sollicitation importatnte de la capacité mémoire d'un composant sur une période de temps relativement courte. Une fois que la ressource est épuisée, l'état de composant est signalé anormal. Nous nous servons d'un script 4.4 d'isolation de processus, illustré dans le tableau 8.7, pour allouer 256 MB de mémoire à chaque VM en contrepartie de limiter la mémoire disponible pour un conteneur à 512 MB.

Tableau 4.4 Commande Docker pour stresser la mémoire

sudo docker run $--rm$ $--name$ stressMem $--memory$ $512m$ stress $--vm 4$ $--vm-bytes$ $256M$ $--timeout$ 10

Les schémas 4.4 et 4.3 décrivent le taux d'utilisation du CPU et de la mémoire lors d'un exemple de surveillance des prestations d'un composant du système sous divers profils d'anomalies (CPU, mémoire).

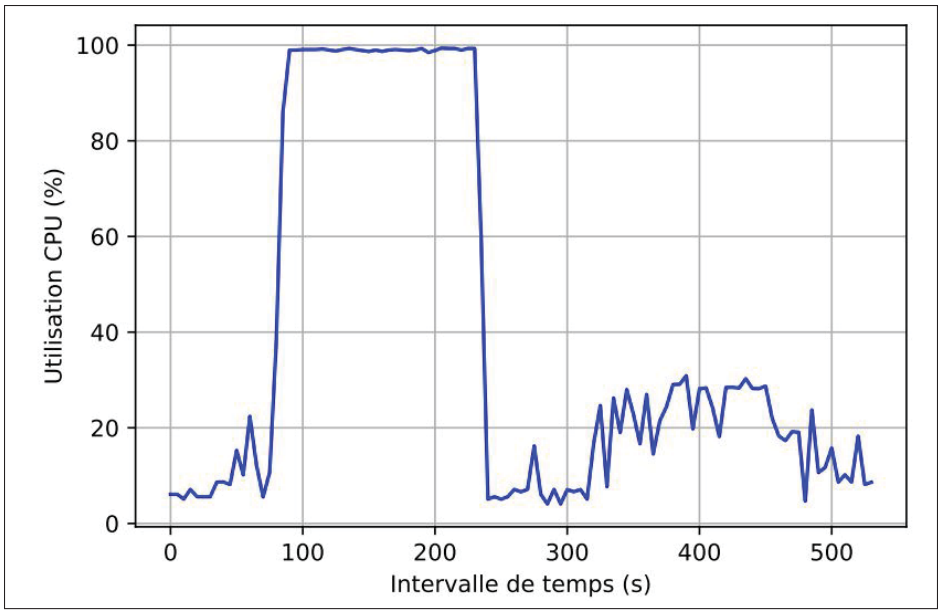


Figure 4.3 Exemple d'évolution de la consommation CPU face à un stress

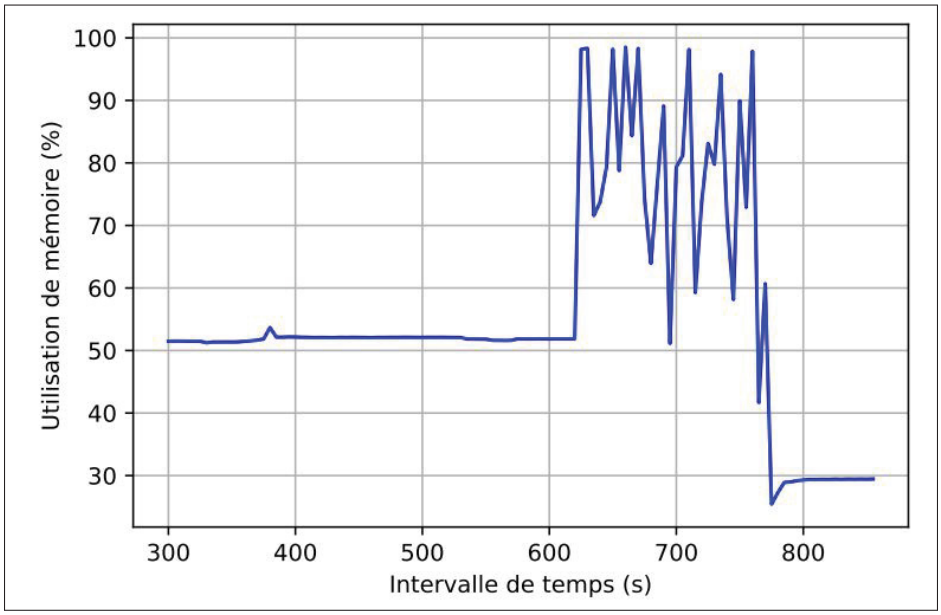


Figure 4.4 Exemple d'évolution de la consommation de mémoire face à un stress

4.4 Plan expérimental

Nos résumons notre plan d'expériemetation dans des figures qui suivent.

Nous commençons tout d'abord en figure 4.5 par le plan d'initialisation qui consiste à appliquer, pour chaque ressource (CPU ou mémoire) une charge initiale nominale. Ceci est fait dans le but de former un ensemble de donnée nominal et évaluer sa dispertion en vue de calculer les seuils limites de consommations correspondantes. Nous exploitons également l'ensemble de donnée nominal pour projeter la structure du réseau bayésien et intilialiser ses paramètres, nottament les distributions des probililités conditionnelles, les taux de confiance et le taux d'atténuation.

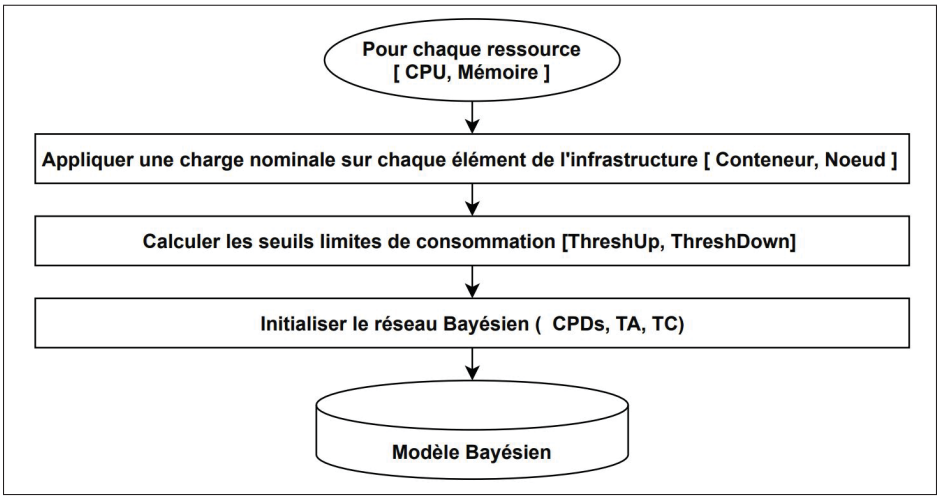


Figure 4.5 Plan d'initialisation

Ensuite, nous nous focalisons sur le plan d'adaptation, décrit en figure 4.6 ou nous allons injecter plusieurs profils d'anomalies sur les éléments de l'infrastructure distribuée. Nous assumons d'appliquer un seul profil à la fois qui peut se propager vers plusieurs nœuds ou conteneurs. Le processus de stress est répété 5 fois pour obtenir les valeurs moyennes des propriétés significatives.

Après avoir injecté des anomalies, nous allons évaluer le processus d'adaptation ou d'apprentissage bayésien identifiant et localisant ces anomalies et en suivant leurs pistes de propagation potentielle. Ce processus est appelé inférence et décrit en figure 4.7 .

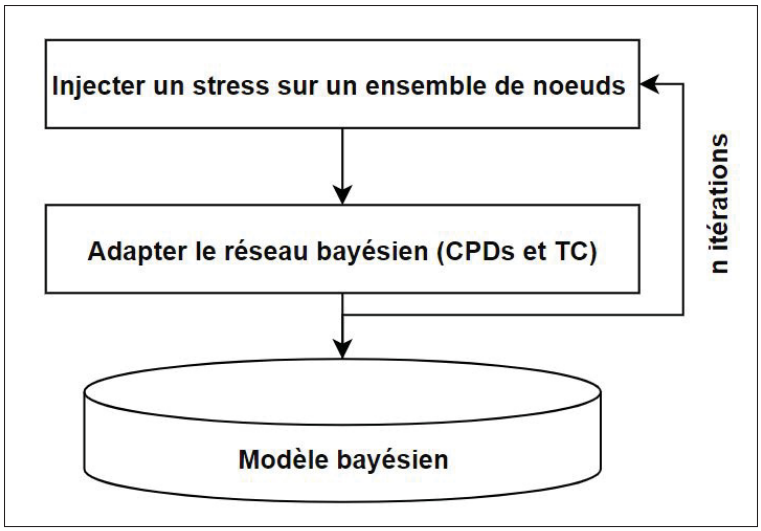


Figure 4.6 Plan d'adaptation

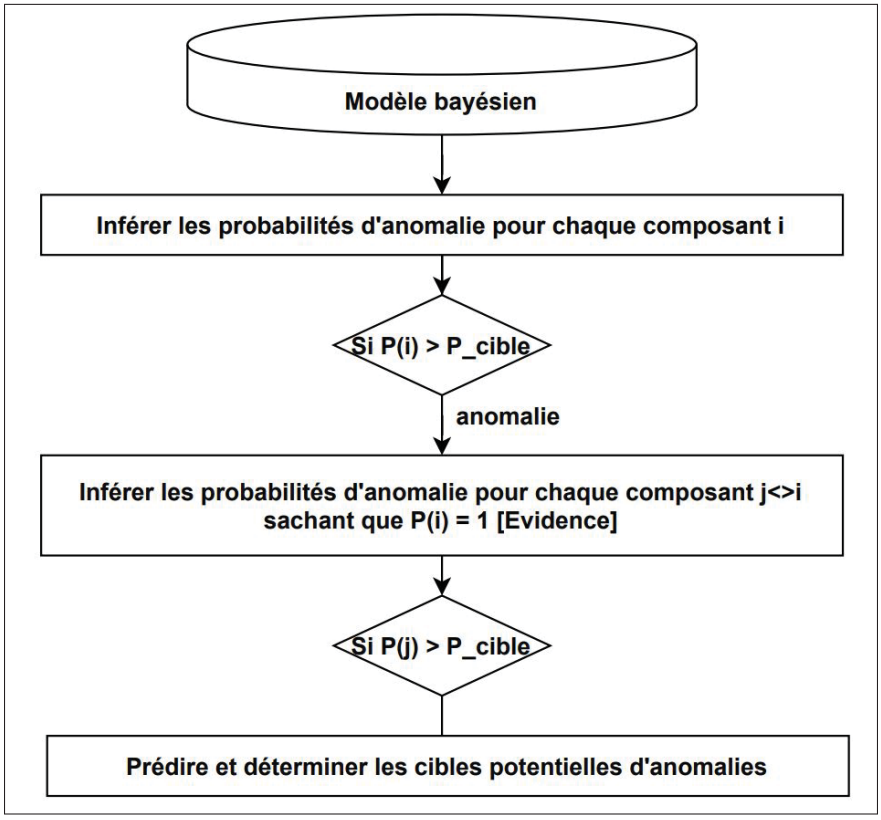


Figure 4.7 Plan d'inférence

Après avoir détecté les anomalies via le franchissement des seuils limites de consommation, nous intervenons proactivement, comme le décrit la figure 4.8, pour prévenir leurs propagations. En effet, nous ajustons les allocations des ressources des cibles prédites ce qui offre une marge de manoeuvre et freine toute piste de propagation d'anomalie.

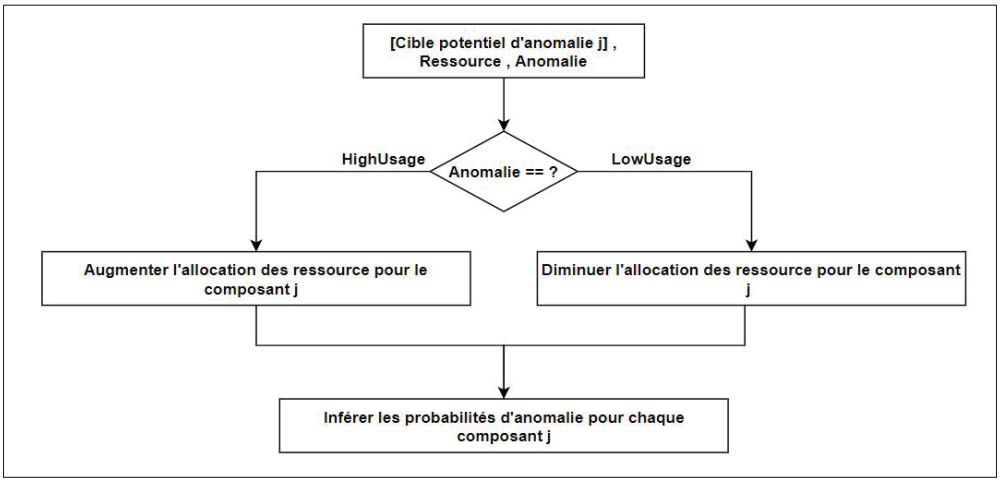


Figure 4.8 Plan de correction

Durant l'expérience, nous discrétisons le temps en intervalles de 5 secondes pour former un base de données d'évaluation d'environ 2160 intervalles répartis sur les 180 minutes d'expérience.

Ce plan expérimental est évalué en matière d'efficacité de ses volets de détection (phase d'adaptation), de prédiction (phase d'inférence) et enfin de récupération des anomalies. Cette efficacité est évaluée durant une application d'une charge stable de requêtes (voisinant les 1000 req/s) de l'utilitaire TPC-W.

4.5 Métriques d'évaluation

Nous entendons par évaluer l'efficacité des phases de détection, d'adaptation et d'inférence de notre approche, la détermination des taux d'anomalies capturées, retracées et récupérées par notre méthode. L'efficacité est examinée en calculant diverses métriques telles que des matrices de confusion et des taux d'erreur. Cette efficacité est investiguée sous plusieurs angles :

– capacité des ressources : cette perspective est motivée par une planification correcte de la capacité d'une ressource à satisfaire les sollicitations d'une charge de travail nominale. Dans ce contexte, l'objectif est de détecter un comportement anormal dont la valeur est supérieure à un seuil prédéfini reflétant les capacités limites disponibles.
– comportement d'une ressource d'un composant : une anomalie intervient comme un signal potentiel de fluctuation indésirable du comportement d'une ressource. Par conséquent, notre approche prévoit la définition des seuils dynamiques de consommation dont leurs franchissement caractérisent directement une détection d'une anomalie.

4.5.1 Matrice de confusion

– Précision : c'est le ratio des anomalies correctement détectées par rapport à la somme des alertes annocées. Une précision élevée indique que le modèle reflète convenablement le comportement de l'infrastructure physique.

$$Precision = \frac{Nombre\ des\ detections\ correctes}{Nombre\ total\ des\ alertes} \tag{4.1}$$

– Rappel : c'est le ratio entre les anomalies correctement détectées et le nombre total des alertes correctement annocées (des vraies anomalies). Un rappel élevé signifie une efficacité du processus de détection (peu de cas d'anomalies ne sont pas détectés).

$$Rappel = \frac{Nombre\ des\ detetcions\ correctes}{Nombre\ total\ des\ anomalies} \tag{4.2}$$

– Score F1 : évalue la fiabilité de la détection en se basant sur les taux de précision et rappel. Ce score est borné entre 0 et 1. Plus que sa valeur s'élève, meilleur sont les résultats.

$$Score\ F1 = \frac{2 * Precision * Rappel}{Precision + Rappel} \tag{4.3}$$

– Taux de fausses alertes (TFA) : caractérise le nombre de fausses alertes, c'est-à-dire les composants déclarés en état d'anomalie par le modèle. Ce taux varie entre 0 et 1, plus que sa valeur est faible plus que le modèle détecte efficacement les anomalies.

$$TFA = \frac{FP}{VN + FP} \tag{4.4}$$

Comme le souligne le tableau 4.5, vrai positif (VP) signifie que la détection d'une anomalie est correcte. De la même manière, les faux positifs (FP) signifient que la détection d'une anomalie est incorrecte car le modèle associe un comportement normal à une anomalie. Faux négatif (FN) signifie que le modèle interprète une anomalie existante comme un comportement normal. Enfin Vrai négatif (VN) signifie que le modèle peut identifier correctement un comportement normal comme normal.

Tableau 4.5 Matrice de confusion

Observé	Détecté	
	Anomalie	Normal
Anomalie	Vrai Positif (VP)	Faux Négatif (FN)
Normal	Faux Positif (FP)	Vrai Négatif

4.5.2 Métriques d'erreur

Le résultat d'une détection est quantifié en comparant la prédiction à l'ensemble de données initiales dans le but de qualifier la précision du modèle. Ceci se fait en évaluant des métriques comme l'erreur quadratique moyenne (RMSE - Root Mean square Error), le score R^2, le taux de prédiction et celui de précision. Ces métriques caractérisent le contraste entre les valeurs prédites et celles observées par le modèle.

– erreur quadratique moyenne (RMSE) : mesure la dispersion entre les valeurs prédites par le modèle et les valeurs observées. Comme l'indique l'équation 4.5, ou x est la sortie observée,

y est la sortie prédite et O_{nb} est le nombre d'observations dans l'ensemble de données, une valeur RMSE plus faible est signe d'efficacité du mécanisme de prédiction.

$$RMSE = \sqrt{\sum_{i=1}^{O_{nb}} (y_i - x_i)^2 / O_{nb}} \tag{4.5}$$

– score R^2 : qualifie la justesse du modèle de prédiction. Une valeur qui tend vers 100% cartérise une bonne fiabilité du modèle tandis qu'une autre valeur qui s'approche de 0% est signe d'incertitude de la justesse du modèle. Nous décrivons ce score en équation 4.6, tel que x est la sortie observée, y est la sortie prédite, Onb est le nombre d'observations dans l'ensemble de données, et $(y_i - z)$ est la variance des deux sorties.

$$R^2 = 1 - (\sum_{i=1}^{O_{nb}} (y_i - x_i)^2 / \sum_{i=1}^{O_{nb}} y_i - z))z \tag{4.6}$$

4.6 Résulats et discussions

Pour adresser les scénarios de propagation de la dégradation de performances, nous injectons des fautes (saturation CPU et fuite de mémoire) sur deux étapes comme le montre la figure 4.9 : au niveau du système (cluster) et au niveau des composants tels que les nœuds et les conteneurs, un composant à la fois.

Nous avons privilégié l'application d'une contrainte au niveau des nœuds pour montrer à la fois la cause (sur le cluster) et l'effet (sur les conteneurs) de cette contrainte sur d'autres éléments dépendants de l'infrastructure. Un délai de pause est appliqué entre chaque faute injectée. Vu la variance des délais d'injections, le temps de détection d'anomalie, de prédiction et de récupération diffèrent d'un composant à l'autre, comme nous le verrons plus loin dans les sous-sections suivantes.

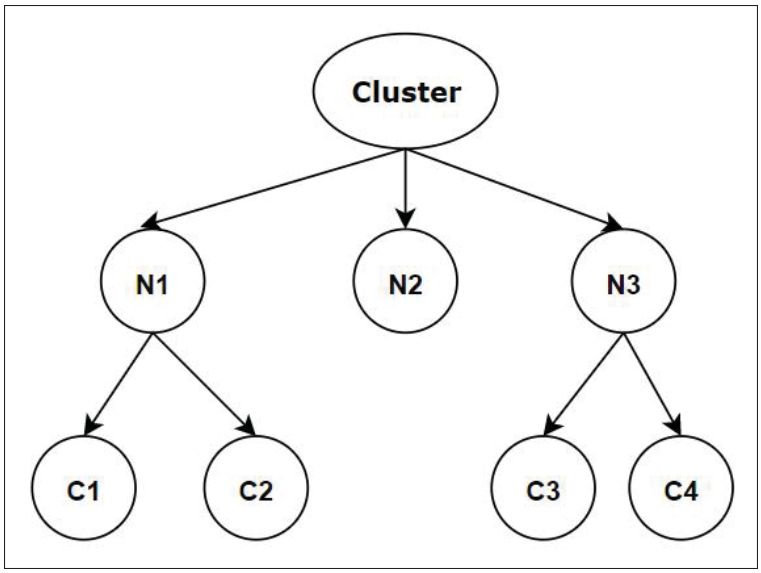


Figure 4.9 Architecture d'expérimentation

4.6.1 Évaluation de la phase de détection

Dans ce cas, nous mettons en question l'efficacité de notre modèle à détecter des actions de franchissement des seuils de consommation de ressources, c'est-à-dire nous évaluons sa capacité d'alerter réactivement la présence d'un comportement anormal au niveau d'un nœud ou un conteneur de l'infrastructure..

Dans un contexte particulier, nous créons un nouveau nœud N4 indépendant des autres nœuds existants, comme le montre la figure 4.10. Nous stressons ses ressources à des instants spécifiques par une variante d'anomalie à la fois. Nous saturons la ressource CPU en premier lieu à l'intervalle 350. Nous détectons un début d'un comportement anormal à l'intervalle 352 (voir figure 4.11) . Nous provoquons en second lieu une fuite de mémoire à l'intervalle 800, détectée par notre modèle à l'intervalle 801 (voir figure 4.12).

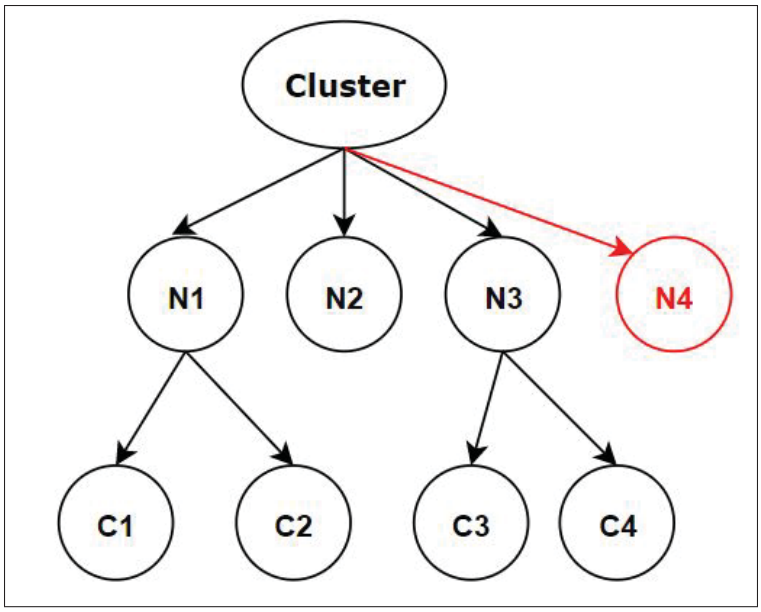


Figure 4.10 Ajout d'un noeud N4

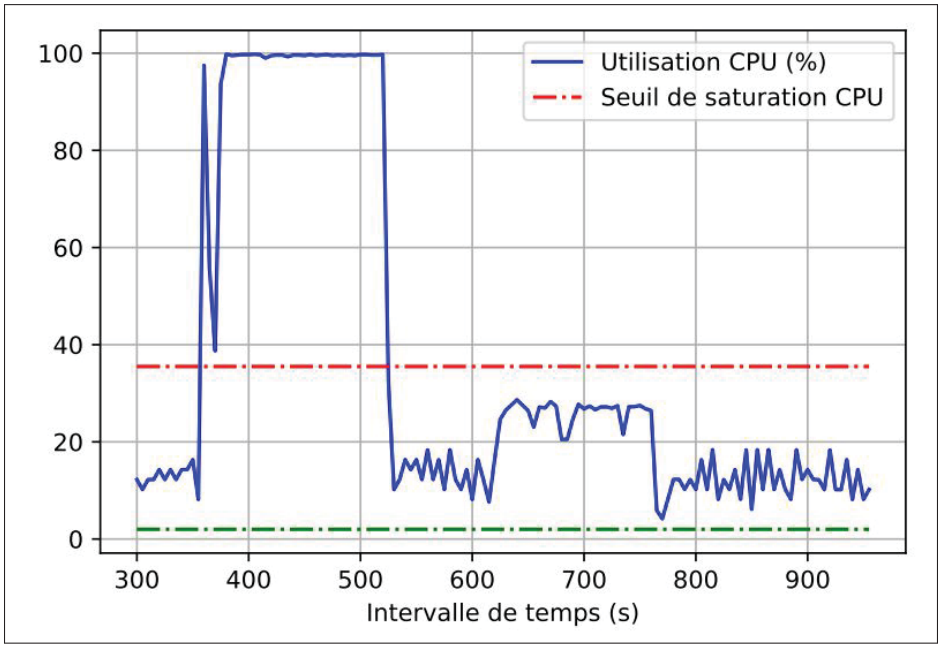


Figure 4.11 Détection d'une saturation CPU au niveau N4

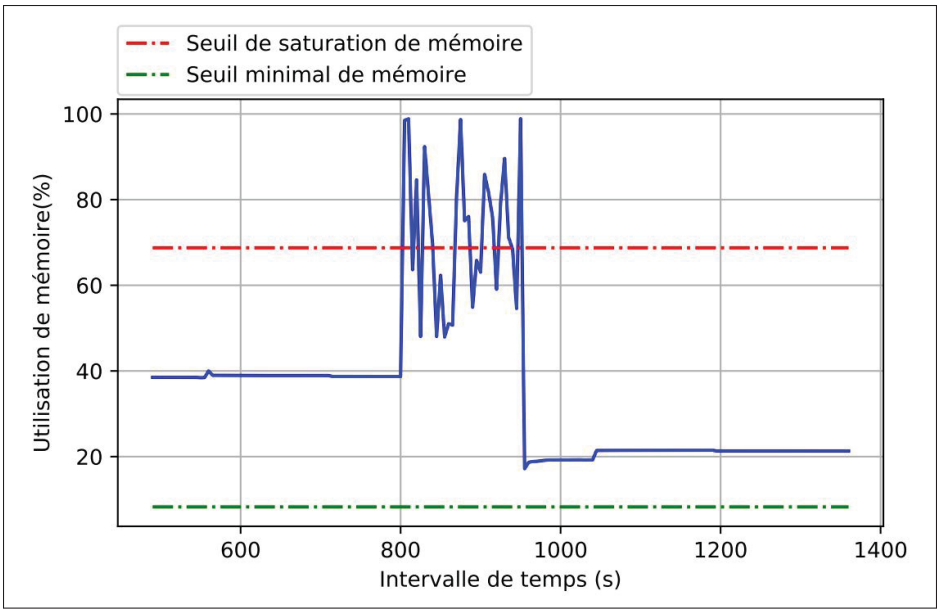


Figure 4.12 Détection d'une fuite de mémoire au niveau N4

Nous reproduisons la même logique au niveau des conteneurs. En effet, nous ajoutons un nouveau conteneur C5 au nœud N1. Ce dernier est indépendant de ses voisins, comme le montre la figure 4.13.

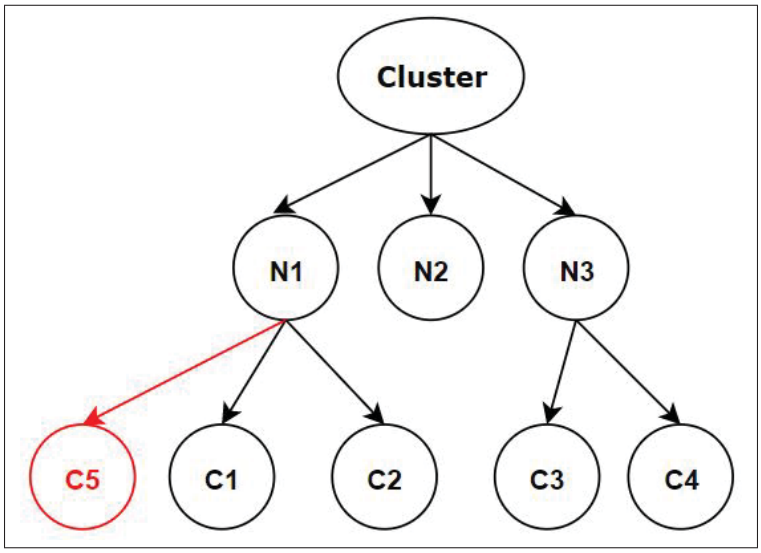


Figure 4.13 Ajout d'un conteneur C5

Nous provoquons une suite d'anomalies successives à savoir une saturation CPU et une fuite de mémoire. À titre indicatif, tel qu'indiquer en figure 4.14, nous notons de stresser la ressource mémoire à l'intervalle 365, tandis que le seuil maximal de consommation mémoire, qualifiant la détection d'une anomalie, est franchi à l'intervalle 366.

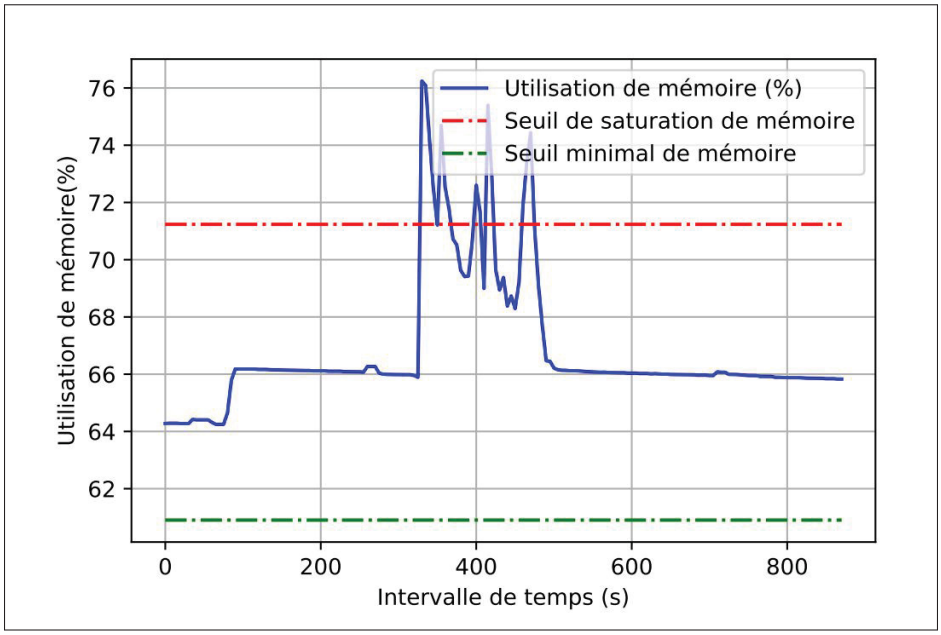


Figure 4.14 Détection d'une fuite de mémoire au niveau C5

Nous généralisons l'évaluation de la performance de détection des anomalies en spécifiant les taux de précision et de rappel réalisé au cours des 2160 intervalles d'expérience . Nous déduisons également le score F1 ainsi que le taux des fausses alertes. Cette évaluation, décrite en tableaux 4.6, révèle des résultats prometteurs comparables à des approches existantes en littérature.

Tableau 4.6 Mesures de performace de la détection

Précision	0.95
Rappel	0.91
Score F1	0.93
Taux de fausses alertes	0.28

4.6.2 Évaluation de la phase de prédiction

Évaluer la prédiction des anomalies de notre approche d, apprentissage bayésien revient à évaluer la valider des scénarios de propagation des anomalies détectés. Nous nous référons à l'erreur quadratique moyenne pour qualifier la dispersion entre les anomalies prédites par notre modèle et les anomalies observées dans la base de données d'évaluation.

Nous ciblons particulièrement la propagation d'une anomalie qui apparaît sur le nœud hébergeant le serveur web TPC-W et ses répercussions directes sur les nœuds hébergeant les applications serveur et client. À cet effet, nous provoquons une fuite de mémoire au niveau du nœud N1 qui occupe le pilier serveur de TPC-W. Vu la forte interaction entre les 3 piliers de cet utilitaire, nous observons, à travers la figure 4.15 des répercussions directes sur la consommation de ressources mémoire au niveau des nœuds occupants respectivement l'application client et la base de données. Par exemple nous notons que pour un scénario de fuite de mémoire provoqué à l'intervalle 550 sur le nœud 1, l'anomalie se propage vers le nœud N3, dit nœud dépendant, vers l'intervalle 607.

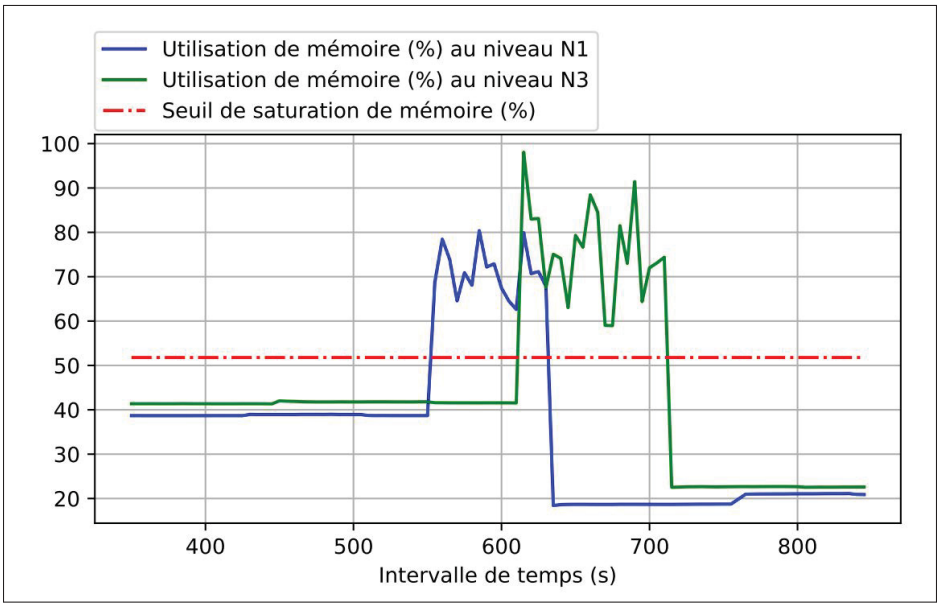


Figure 4.15 Propagation d'une fuite de mémoire de N1 à N3

Nous caractérisions et confirmons notre observation en figure 4.15 en investiguant l'évolution de la probabilité de prédiction d'anomalie au niveau N3. Notre modèle bayésien adaptatif proposé, confirme à travers la figure 4.16 une forte corrélation, dite dépendance causale, entre les nœuds N1 et N3. En effet, nous voyons que durant l'application du stress au niveau N1, la probabilité de prédiction d'une anomalie du même genre au niveau N3 augmente graduellement. Si nous caractérisons cette probabilité comme facteur qualitatif et non quantitatif, autrement dit nous nous intéressons plutôt au changement brusque de la probabilité d'anomalie que sa valeur, nous prédisons proactivement une anomalie potentielle au niveau N3 à partir de l'intervalle 597.

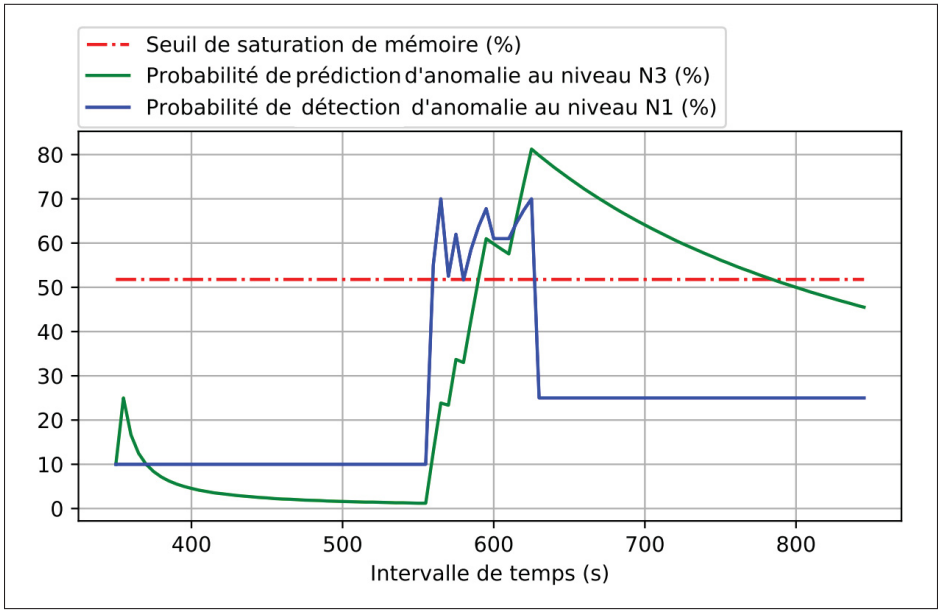


Figure 4.16 Évolution de la probabilité de prédiction d'anomalie au niveau N3

Cependant, si notre modèle d'adaptation bayésiena qualifié la propagation d'une anomalie du nœud N1 vers N3, nous projetons sa capacité à caractériser une relation d'indépendance fonctionnelle comme celle entre les nœuds N1 et N4. Les figures 4.17 et 4.18 décrivent la non-propagation d'un stress de la ressource mémoire appliqué sur N1 vers N4. Cette indépendance est illustrée en figure 5.19 qui évalue la modélisation bayésienne de la dégradation de prédiction d'une anomalie au niveau N4.

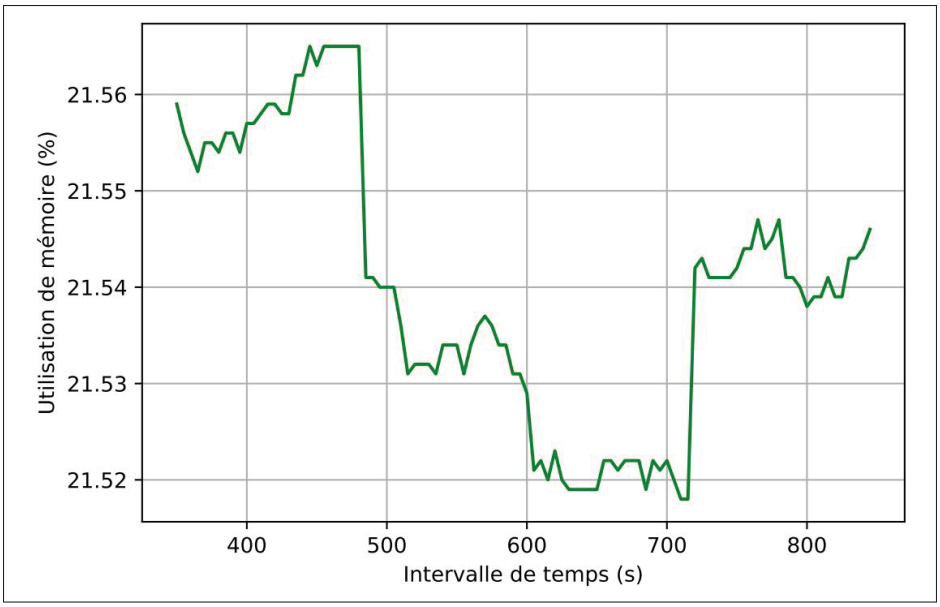


Figure 4.17 Utilisation de mémoire au niveau N4

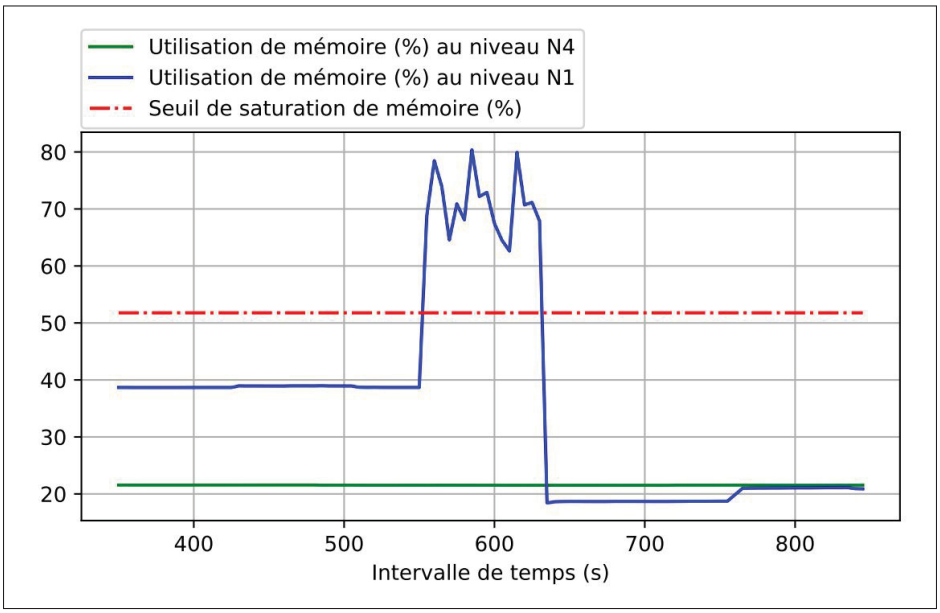


Figure 4.18 Effet d'une fuite de mémoire au niveau N1 sur N4

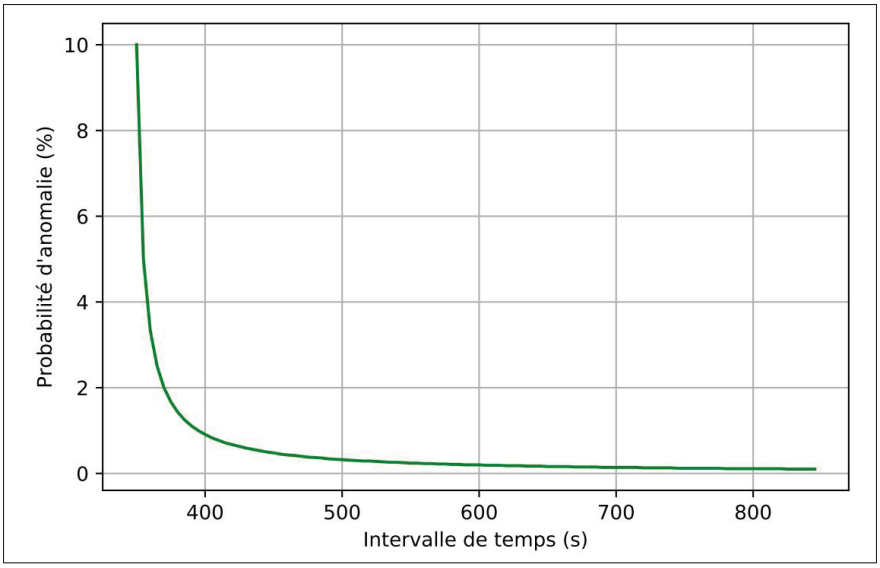


Figure 4.19 Évolution de la probabilité de prédiction d'anomalie au niveau N4

Nous reproduisons la même logique mentionnée ci-haut au niveau des conteneurs. En effet, nous évaluons la capacité de notre modèle bayésien à retracer la propagation d'une saturation CPU appliqué sur le conteneur C1 vers les autres conteneurs de l'infrastructure.

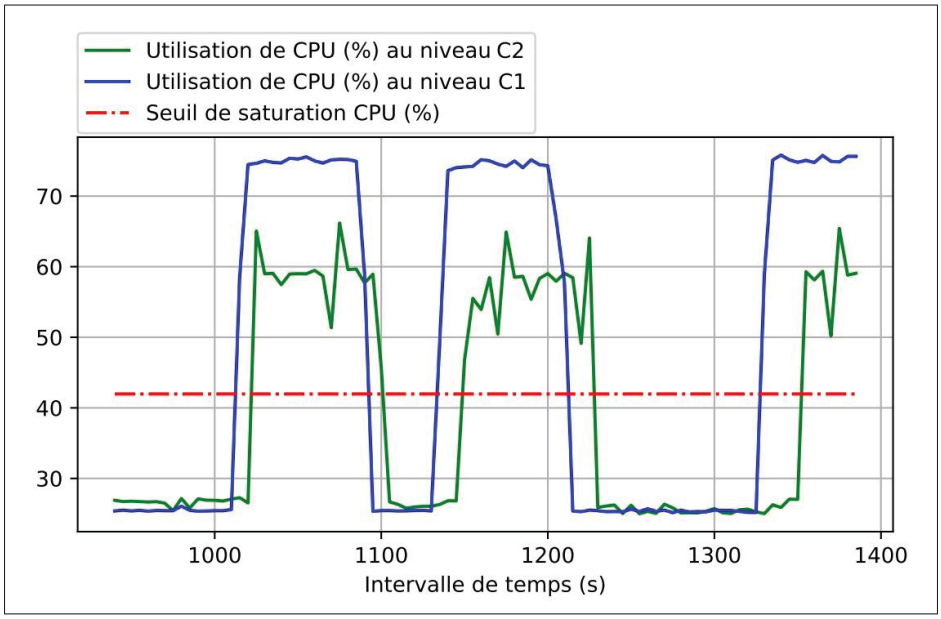


Figure 4.20 Effet d'une saturation CPU au niveau C1 sur C2

Nous observons d'une part à travers la figure 4.20 une relation de dépendance (cause-conséquence) menant à une propagation de l'anomalie vers le conteneur C2. Nous notons sur la figure 4.21 l'évolution de la prédiction de cette anomalie qui augmente graduellement suite à plusieurs tentatives de stress sur C1.

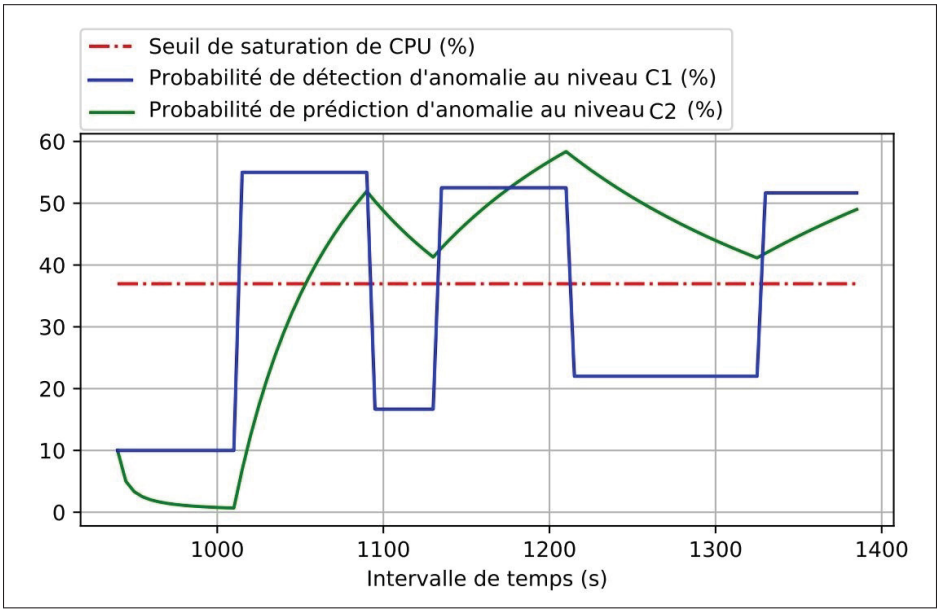


Figure 4.21 Évolution de la probabilité de prédiction d'anomalie au niveau C2

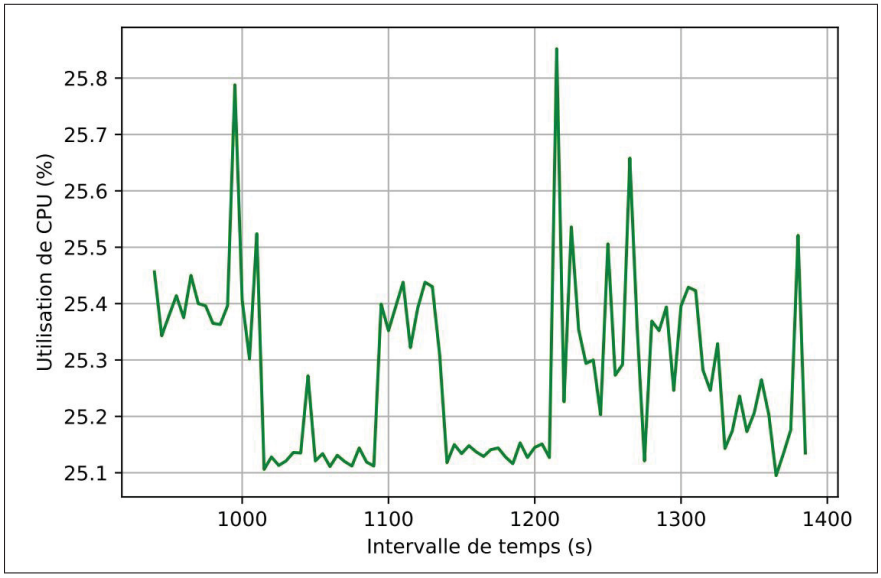


Figure 4.22 Utilisation de CPU au niveau C5

D'autre part, nous assertons la capacité de notre approche à expliciter un scénario d'indépendance fonctionnelle, comme celui-ci entre C1 et C5. En effet, nous observons en figure 4.22 et 4.23 que l'anomalie engendrée sur C1 ne se propage pas vers C5 qui continue à opérer sous un régime nominal. Ce fait est confirmé en suivant sur la figure 4.24 la dégradation du taux de prédiction d'anomalie au niveau C5.

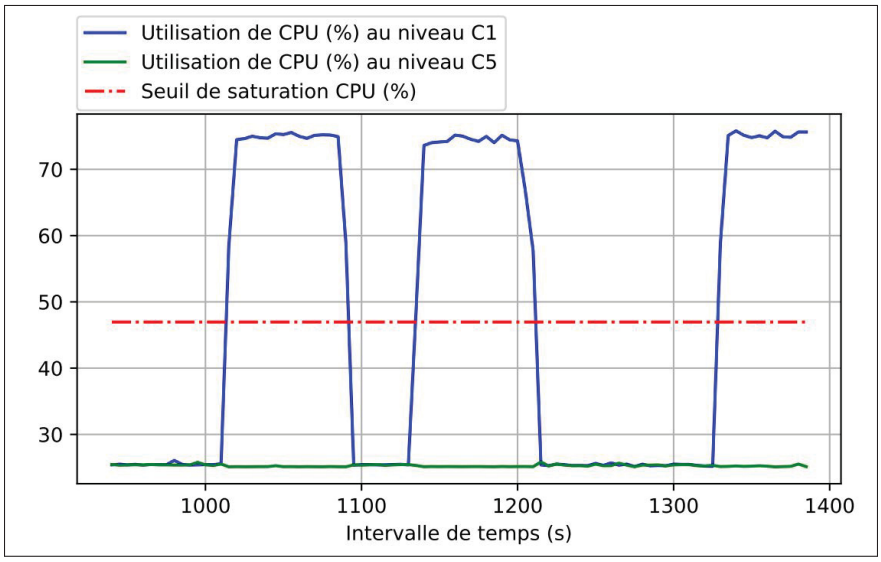


Figure 4.23 Effet d'une saturation CPU au niveau C1 sur C5

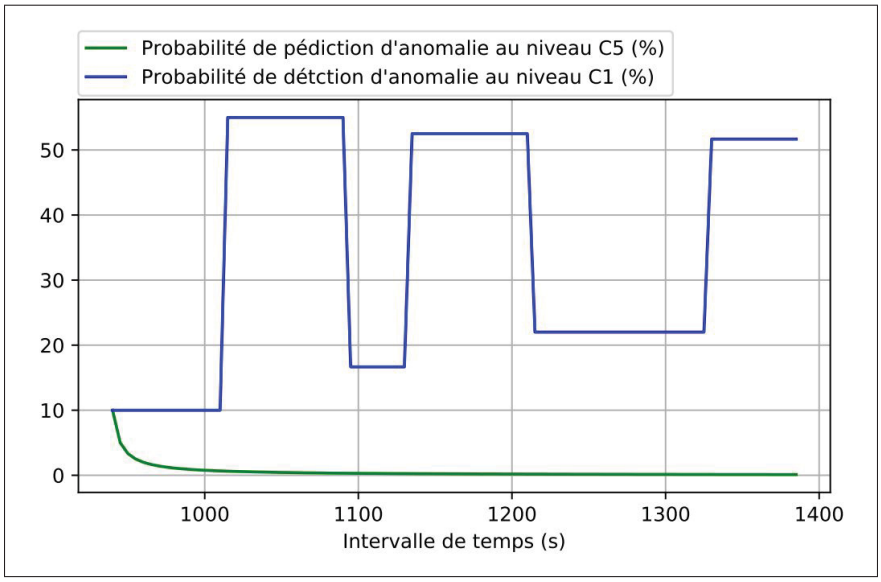


Figure 4.24 Évolution de la probabilité de prédiction d'anomalie au niveau C5

Nous généralisons la validité de ces cas particuliers mentionnés ci-dessus en étalant notre évaluation de la performance de prédiction des anomalies au cours des 2160 intervalles d'expérience, dont un tiers de ces intervalles représentent la phase d'initialisation et les deux autres tiers présentent la phase d'adaptation ou d'apprentissage bayésien.

Nous soulignons, en comparant les résultats inscrits en tableau 4.7, une amélioration significative de la performance de prédiction entre la phase d'initialisation du modèle bayésien et la phase d'adaptation. Cette caractéristique est inspectée dès les premières phases de notre projet. En effet, nous avons quantifié la pente de montée (entre les intervalles 180 et 320) de la courbe de prédiction d'anomalie au cours d'une phase d'initialisation en figure 4.25 comparativement à celle en phase d'adaptation présente en figure 4.26. Nous avons constaté que la fiabilité de notre modèle augmente au fil des opérations d'adaptation des paramètres.

Tableau 4.7 Mesures de performace de la prédiction

	Phase d'initialisation	**Phase d'adaptation**
RMSE	0.25	0.16
$\boldsymbol{R^2}$	76.15%	89.61%

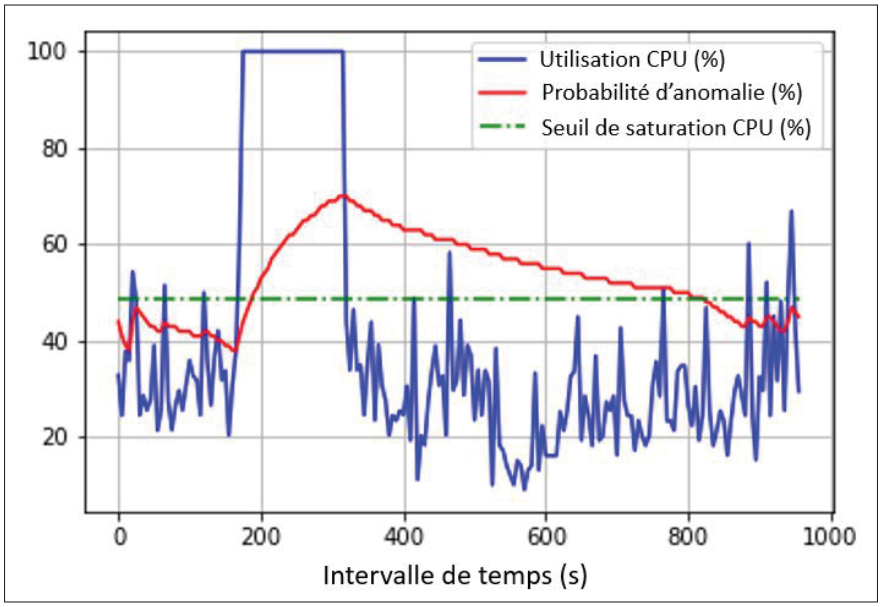


Figure 4.25 Performance en phase d'initialisation

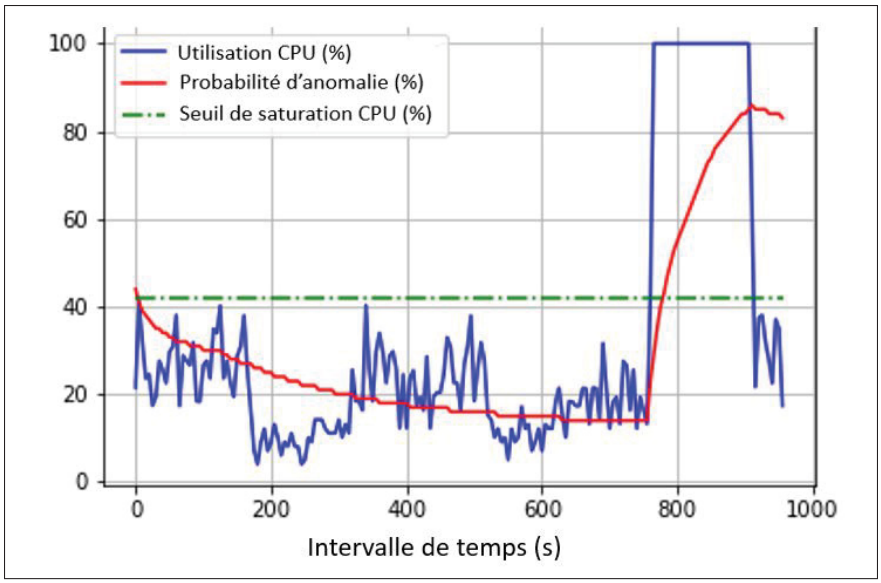


Figure 4.26 Performance en phase de prédiction

Nous soulignons également, l'efficacité de notre approche bayésienne pour traiter les pics d'utilisation anormale des ressources (indiquées à 820s et 930s en figure 4.27) en ne les considérant pas comme des anomalies. Cette capacité est absente avec les approches réactives à base de règles de décision.

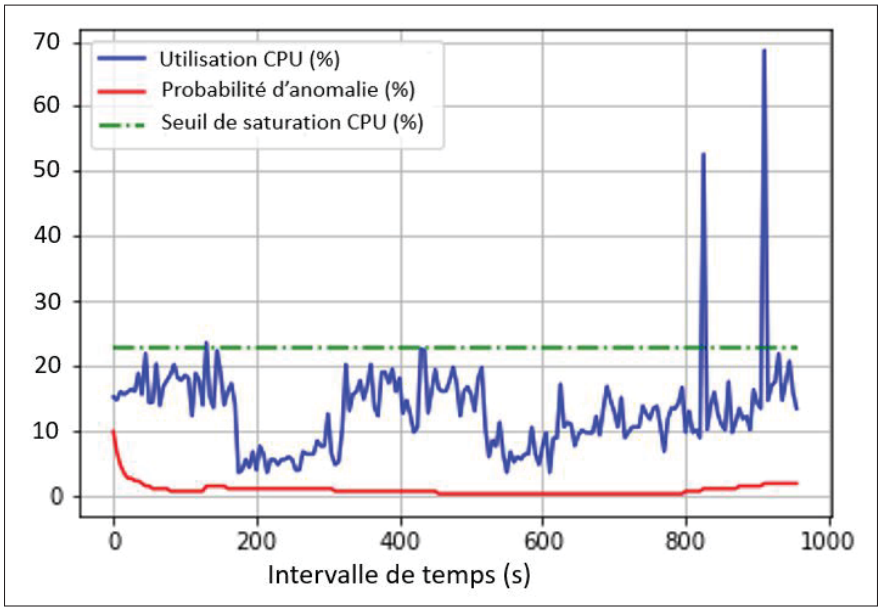


Figure 4.27 Gestion des pics de consommation

4.6.3 Évaluation de la phase de récupération

L'élaboration de la phase récupération ne fait partie des objectifs principaux de notre projet cependant dans la quête de s'aligner avec à le formaliser MAPE-K, nous avons évalué le volet correction de l'approche par le biais d'un scénario qui s'applique au niveau des conteneurs.

En effet nous avons stressé la ressource CPU au niveau C1. Notre modèle d'adaptation bayésien a inféré une propagation potentielle d'anomalie vers le conteneur C2. Nous observons en figure 4.29, qu'en absence d'une action de correction proactive à appliquer sur le conteneur C2, le taux de prédiction d'anomalie augmente en suivant la persistance du stress au niveau C1.

Notre action corrective était le recours vers une augmentation du quota des ressources CPU offertes le nœud N1 pour le conteneur C2 sans se contrarier de redéployer infrastructure physique ou reconfigurer la modélisation bayésienne. Suite à cette action proactive nous observons, sur la figure 4.28, la récupération de l'anomalie et le retour au régime nominal d'opération par le conteneur C2 sans se soucier de la continuité de l'effet stress sur C1. L'incorporation de cette mesure proactive a pour effet de redescendre le taux de prédiction d'anomalie, comme l'illustre la figure 4.29 à partir de l'intervalle 1395.

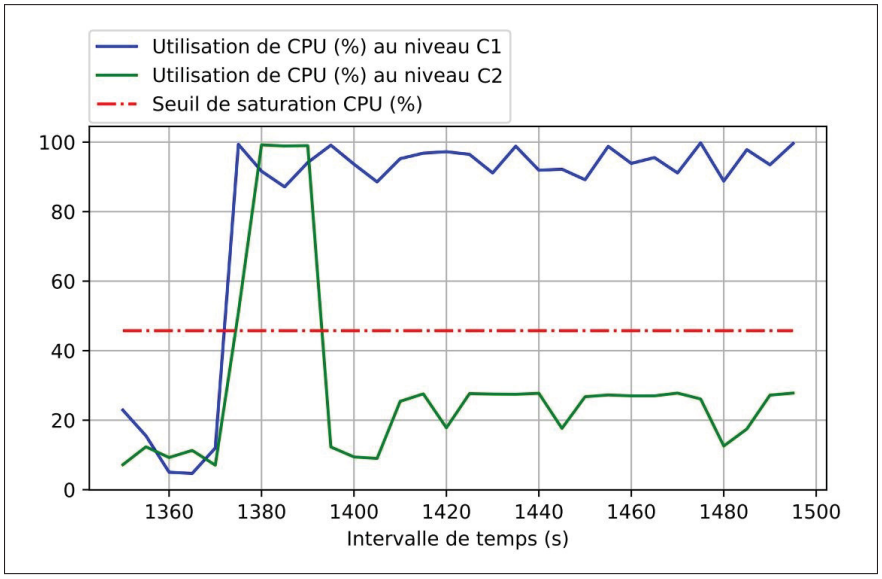


Figure 4.28 Récupération d'anomalie au niveau C2

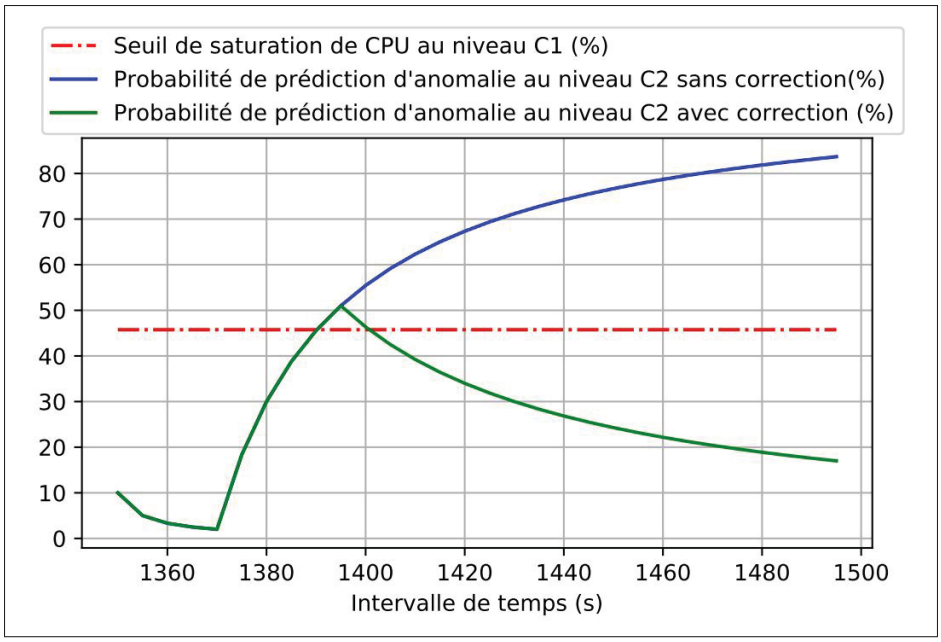


Figure 4.29 Comparaison de l'application d'une action corrective

Pour généraliser notre évaluation nous avons mesuré :

– la précision de la procédure de récupération, c'est le rapport entre le nombre d'anomalies rétablies et celui des anomalies détectées.
– le délai de rétablissement, c'est la durée moyenne que prend l'architecture pour rétablir la propagation dun comportement anormal à partir de la prédiction de l'anomalie jusqu'à sa récupération.

Les résultats du tableau 4.8, montrent qu'une fois que le modèle bayésien adaptatif aborde sa phase de fiabilité, il peut efficacement retracer et récupérer proactivement des anomalies de saturation des ressources avec une précision proche des 86%. En effet nous avons ciblé 21 anomalies prédites correctement et réparties sur la liste des conteneurs de l'architecture à des délais variant tout au long des 2160 intervalles de notre expérimentation.

Tableau 4.8 Mesures de performace de la récupération

Précision	86%
Délai moyen de récupération	120 secondes

Parmi ces cibles, nous avons réussi à prévenir 18 et ainsi limiter la dégradation des performances des conteneurs en question. Par ailleurs, il est important de noter que la validité de ces résultats se restreint pour les anomalies apparues au niveau des conteneurs.

4.7 Conclusion

Ce chapitre expose une évaluation de l'architecture proposée sous plusieurs angles.

D'abord, multiples anomalies sont injectées à différents niveaux du système pour stresser les ressources de calcul des composants. Nous nous contentons de surveiller les niveaux d'utilisation de CPU et de la mémoire sous un régime de travail nominal assuré par l'utilitaire TPC-W et contrarié via divers scénarios de stress.

Ensuite, pour chaque phase de l'approche bayésien ne adaptative proposée, (détection, prédiction et récupération), nous exposons et discutons les résultats réalisés.

Pour la phase de détection, où nous nous sommes intrigués sur la capacité de notre solution à alerter réactivement la présence d'un comportement anormal au niveau d'un nœud ou un conteneur de l'infrastructure, les résultats révèlent un taux de précision de 0.95.

Quant à la phase de prédiction, nous avons questionné la validité des scénarios de propagation des anomalies détectés. Nous nous sommes référés à l'erreur quadratique moyenne pour qualifier la dispersion entre les anomalies prédites par notre modèle et les anomalies observées dans la base de données d'évaluation. En effet le jeu de données généré est réparti entre les processus d'initialisation et d'adaptation. Nous avons constaté une amélioration significative de la performance de prédiction entre la phase d'initialisation du modèle bayésien et sa phase

d'adaptation. Les résultats révèlent que le modèle est capable de retracer une propagation d'anomalie avec un taux de précision proche des 90%.

En plus, et comme suite logique de la phase de prédiction, nous avons évalué des scénarios spécifiques de récupération des anomalies prédites qui ont démontré qu'une fois que le modèle est configuré correctement, il peut intervenir proactivement et recouvrir des anomalies avec une précision de 86% mais avec des délais de rétablissent qui ne s'alignent pas avec des attentes d'un système à flux continu d'opérations ce qui ouvre la voie à des pistes d'amélioration de la solution proposée.

Enfin, nous pouvons conclure que notre modèle d'apprentissage bayésien adaptatif affiche des résultats prometteurs sous diverses sécanios d'expérimentation et différents profils anomalies.

CONCLUSION ET RECOMMANDATIONS

La gestion d'un système distribué représente une opération cruciale en matière de performance, d'efficacité et de fiabilité. Pour ce faire, nous avons proposé une approche statistique adaptative capable de détecter, en se basant sur un suivi des niveaux d'utilisation des ressources, des éventements de dysfonctionnements au niveau des composants du système et retracer, voire prédire les pistes possibles de propagation de ces anomalies vers d'autres éléments de l'infrastructure. Pour ce faire, nous avons adopté un formalisme bayésien pour :

– détecter un dysfonctionnement surgissant sur un composant du système.
– retracer et prédire les pistes potentielles de propagation des anomalies.
– procéder à une action de récupération proactive en guise de prévenir la propagation des anomalies.

Nous avons mis en œuvre l'approche proposée dans un environnement informatique conteneurisé. L'architecture globale intègre des couches de plusieurs nœuds et conteneurs. Chaque nœud contient un ou plusieurs conteneurs qui appartiennent à un cluster. Nous avons recueilli des observations à partir des nœuds et des conteneurs pour nous en servir ultérieurement lors des opérations de détection et de prédiction des anomalies.

La démarche considérée s'aligne avec le formalise d'autocontrôle MAPE-K, considéré comme modèle de contrôle de référence pour les systèmes adaptatifs. En effet, un système adaptatif est un système qui surveille en permanence ses variations et met à jour les paramètres de ses modèles, qui sont analysés en cours d'exécution pour déceler toute violation des exigences. L'adaptation est déclenchée par la détection de dysfonctionnements ou la prédiction de déviations futures des exigences fonctionnelles. Ainsi, nous avons prévu une phase de détection qui repère l'existence d'anomalies en collectant des données sur l'état d'utilisation des ressources CPU et mémoire au niveau des composants du système. La phase de détection est suivie d'une phase de prédiction

responsable de retracer les pistes potentilles de propagation de l'anomalie vers d'autres éléments de l'infrastructure.

La performance de l'architecture adaptative bayésienne est évaluée en appliquant différents scénarios de tests. Pour la phase de détection, nous avons inspecté la capacité de notre solution à alerter réactivement la présence d'un comportement anormal au niveau d'un nœud ou un conteneur de l'infrastructure. Les résultats ont révélé un taux de précision de 0.95. Quant à la phase de prédiction, nous avons questionné la validité des scénarios de propagation des anomalies détectés. Nous avons qualifié la dispersion entre les anomalies prédites par notre modèle et les anomalies observées dans la base de données d'évaluation. Nous avons constaté une amélioration significative de la performance de prédiction entre la phase d'initialisation du modèle bayésien et sa phase d'adaptation. Les résultats ont démontré que le modèle est capable de retracer une propagation d'anomalie avec un taux de précision proche des 90%. Nous suivons la même logique, nous avons évalué des scénarios spécifiques de récupération des anomalies prédites qui ont démontré qu'une fois que le modèle est configuré correctement, il peut intervenir proactivement et recouvrir des anomalies avec une précision de 86%

Nous concluons que notre modèle d'apprentissage bayésien adaptatif affiche des résultats prometteurs sous divers scénarios d'expérimentations et différents profils anomalies. Cependant, en guise de perspectives futures, nous projetons d'améliorer notre travail en :

– mise en place d'un environnement d'évaluation plus complexe, il serait intéressant d'évaluer un environnement plus complexe sur une plus longue période avec un ensemble plus large de variables afin d'obtenir des résultats plus fiables. Nous prévoyons donc de valider l'approche proposée sur plusieurs jeux de données, de surveiller d'autres métriques tels que le réseau, le temps de réponse et d'injecter plusieurs anomalies simultanément à différents niveaux du système pour évaluer l'architecture face à plusieurs scénarios assez proches de la réalité.

- consolider la phase de récupération, d'autres expériences seront menées pour classer et évaluer des actions de récupération, telles que la répartition ou l'équilibrage des charges pour un conteneur ou un nœud, la création ou résiliation d'un nœud, la division d'un nœud ou conteneur et la migration d'un nœud.
- projeter l'approche en environnement Edge Computing, où nous remettrons en question l'utilisation des ressources de notre approche face à des limitations de consommation imposées par les compostants du Edge.

BIBLIOGRAPHIE

Alcaraz, J., Kaanich, M. & Sauvanaud, C. (2016). *Anomaly detection in cloud applications*. Université Toulouse 1 Capitole.

Avizienis, A., Laprie, J. C. & Randell, B. (2001). *Fundamental concepts of dependability*. Technical Report Series university of Newcastle Upon Tyne Computing Science.

Bali, A., Al-Osta, M., Ben Dahsen, S. & Gherbi, A. (2020). Rule based auto-scalability of IoT services for efficient edge device resource utilization. *Ambient Intelligence and Humanized Computing*, 4-10.

Becker, M., Luckey, M. & Becker, S. (2013). Performance analysis of self-adaptive systems for requirements validation at design-time. *9th international ACM Sigsoft conference on Quality of software architectures (QoSA)*, pp. 43-52.

Chalapathy, R., Menon, A. K. & Chawla, S. (2018). Anomaly Detection using One-Class Neural Networks. *ArXiv abs/1802.06360*, 2–12.

Düllmann, T. F. (2016). *Performance anomaly detection in microservice architectures under continuous change*. (Mémoire de maîtrise, University of Stuttgart, Stuttgart).

Farshchi, M., Schneider, J. G., Weber, I. & Grundy, J. (2016). Anomaly detection of cloud application operations using log and cloud metric correlation analysis. *26th International Symposium on Software Reliability Engineering, (ISSRE)*, pp. 24-34.

Ghaith, S., Wang, M., Perry, P., Jiang, Z. M., O'Sullivan, P. & Murphy, J. (2016). Anomaly detection in performance regression testing by transaction profile estimation. *Software Testing Verification and Reliability*, 26(1), 4-39.

Guan, Q., Chiu, C.-C. & Fu, S. (2016). Cda : A cloud dependability analysis framework for characterizing system dependability in cloud computing infrastructures. *18th Pacific Rim International Symposium on Dependable Computing, (PRDC)*, pp. 11–20.

Heckerman, D. (2008). A Tutorial on Learning with Bayesian Networks. Dans Holmes, D. E. & Jain, L. C. (Éds.), *Innovations in Bayesian Networks. Studies in Computational Intelligence* (vol. 156, pp. 33-82). Berlin, Heidelberg : Springer.

Iber, J., Rauter, T., Krisper, M. & Kreiner, C. (2017). Systems, software and services process improvement. *European Conference on Software Process Improvement, (SPI)*, pp. 150-161.

Ibidunmoye, O., Hernández-Rodriguez, F. & Elmroth, E. (2015). Performance anomaly detection and bottleneck identification. *ACM Computing Surveys*, 48(1), 1-35.

Ibidunmoye, O., Metsch, T. & Elmroth, E. (2016). Real-time detection of performance anomalies for cloud services. *IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pp. 1-2.

Jung, G., Swint, G. S., Parekh, J., Pu, C. & Sahai, A. (2006). Detecting bottleneck in n-tier it applications through analysis. *International Workshop on Distributed Systems : Operations and Management, (DSOM)*, pp. 149-160.

Kratzke, N. (2017). About microservices, containers and their underestimated impact on network performance. *CoRR, abs/1710.0*, 1-5.

Maiga, A., Ali, N., Bhattacharya, N., Sabane, A., Gueheneuc, Y.-G. & Aimeur, E. (2012). Smurf : A svm-based incremental anti-pattern detection approach. *19th Working Conference on Reverse Engineering (WCRE)*, pp. 466-475.

Malkowski, S., Hedwig, M. & Pu, C. (2009). Experimental evaluation of n-tier systems : Observation and analysis of multi-bottlenecks. *International Symposium on Workload Characterization, (IISWC)*, pp. 118-127.

Mariani, L., Monni, C., Pezze, M., Riganelli, O. & Xin, R. (2018). Localizing faults in cloud systems. *11th International Conference on Software Testing, Verification and Validation, (ICST)*, pp. 262–273.

Nistor, A., Song, L., Marinov, D. & Lu, S. (2013). Toddler : Detecting performance problems via similar memory-access patterns. *International Conference on Software Engineering, (ICSE)*, pp. 562–571.

Raj, P. & Raman, A. (2018). Continuous validation of performance test workloads. *Automated multi-cloud operations and container orchestration*, 185-218.

Rastogi, N. & Hendler, J. (2017). Graph analytics for anomaly detection in homogeneous wireless networks - a simulation approach. *arXiv preprint arXiv :1701.06823*, 2-3.

Samir, A. & Pahl, C. (2019a). Detecting and predicting anomalies for edge cluster environments using hidden markov models. *4th IEEE International Conference on Fog and Mobile Edge Computing, (FMEC)*, pp. 21-28.

Samir, A. & Pahl, C. (2019b). Dla : Detecting and localizing anomalies in containerized microservice architectures using markov models. *7th International Conference on Future Internet of Things and Cloud, (FiCloud)*.

Samir, A. & Pahl, C. (2019c). Self-adaptive healing for containerized cluster architectures with hidden markov models. *4th IEEE International Conference on Fog and Mobile Edge*

Computing, (FMEC), pp. 68-73.

Sauvanaud, C., Kaâniche, M., Kanoun, K., Lazri, K. & Silvestre, G. D. S. (2018). Anomaly detection and diagnosis for cloud services : Practical experiments and lessons learned. *Systems and Software*, 139, 84–106.

Sorkunlu, N., Chandola, V. & Patra, A. (2017). Tracking system behavior from resource usage data. *International Conference on Cluster Computing, (ICCC)*, pp. 410-418.

Stephenson, T. A. (2000). *An introduction to bayesian network theory and usage*. (Mémoire de maîtrise, IDIAP Research Report).

Syer, M. D., Shang, W., Jiang, Z. M. & Hassan, A. E. (2017). Continuous validation of performance test workloads. *Automated Software Engineering*, 24(1), 189-231.

Tan, Y., Nguyen, H., Shen, Z., Gu, X., Venkatramani, C. & Rajan, D. (2012). PREPARE : Predictive Performance Anomaly Prevention for Virtualized Cloud Systems. *International Conference on Distributed Computing Systems*, pp. 285–294.

Waller, J., Ehmke, N. C. & Hasselbring, W. (2015). Including performance benchmarks into continuous integration to enable devops. *ACM SIGSOFT Software Engineering Notes*, 40(2), 1-4.

Wang, T., Xu, J., Zhang, W., Gu, Z. & Zhong, H. (2018). Self-adaptive cloud monitoring with online anomaly detection. *Future Generation Computer Systems*, 80, 89–101.

Wert, A. (2013). Performance problem diagnostics by systematic experimentation. *18th international doctoral symposium on Components and architecture*, pp. 1-6.

Wert, A. (2015). *Performance problem diagnostics by systematic experimentation*. (Thèse de doctorat, Karlsruher Instituts für Technologie).

Yan, D., Xu, G. & Rountev, A. (2012). Uncovering performance problems in Java applications with reference propagation profiling. *34th International Conference on Software Engineering, (ICSE)*, pp. 134-144.