Prédiction dynamique de la consommation de ressources dans des environnements virtualisés

par

Souhila BENMAKRELOUF

THÈSE PAR ARTICLES PRÉSENTÉE À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE COMME EXIGENCE PARTIELLE À L'OBTENTION DU DOCTORAT EN GÉNIE
Ph. D.

MONTRÉAL, LE 16 DÉCEMBRE 2020

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CETTE THÈSE A ÉTÉ ÉVALUÉE

PAR UN JURY COMPOSÉ DE :

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

M. Chakib Tadj, président du jury
Département de génie électrique, l'École de technologie supérieure

Mme Halima Elbiaze, examinatrice externe
Département d'informatique, UQAM

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

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

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

LE 7 DÉCEMBRE 2020

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens tout particulièrement à exprimer ma gratitude à Mme Nadjia Kara, la directrice de cette thèse, pour ses encouragements, ses conseils et son écoute. Je tiens, également, à lui exprimer toute ma reconnaissance pour son soutien constant et sa générosité tout au long de cette thèse. Son aide inestimable a permis à ce travail de voir le jour. Mme Kara, c'était un honneur et un privilège de travailler avec vous.

J'adresse mes sincères remerciements à M. Chakib Tadj, Mme Halima ELbiaze, M. Chamseddine Talhi et M. Aris Leivadeas pour avoir accepté d'évaluer ce travail ainsi que pour leurs conseils et leurs orientations.

J'exprime ma gratitude à M. Rafi Rabipour, M. Claes Edstrom et M. Yves Lemieux de la compagnie Ericsson Canada, ainsi que M. Jean-Yves Bernard de la compagnie Rogers pour leurs judicieux conseils et leurs contributions pour la réalisation de ce travail.

Je tiens également à remercier mes collègues chercheurs pour leur support et leur gentillesse. Notre collaboration tout au long de ces années m'a beaucoup appris.

Je ne remercierai jamais assez mes très chers parents, qui m'ont toujours entouré de leur amour, leur bienveillance et leur soutien. La force, le courage et la résilience de ma mère durant les épreuves de la vie m'ont donnés la force de persévérer. Je lui dois tout ce que j'ai accompli dans ma vie.

Je remercie mes sœurs et mes frères pour leur soutien, leur encouragement et leur aide.

Enfin, je tiens à exprimer ma reconnaissance envers mes amies, pour leur sincère amitié, leurs conseils et leur support.

Merci à toute personne qui a participé de près ou de loin à ce travail

Prédiction dynamique de la consommation de ressources dans des environnements virtualisés

Souhila BENMAKRELOUF

RÉSUMÉ

Les environnements infonuagiques sont en perpétuelle évolution et offrent de plus en plus d'opportunités pour développer et héberger de nouvelles applications et services. Dans de tels environnements, la gestion efficace des ressources représente un des nombreux et importants défis à relever pour respecter les contrats de service (*Service Level Agreement* : *SLA*) négociés pour y héberger ces applications. En effet, la variabilité et l'imprévisibilité des charges de trafic que pourraient imposer ces applications/services et les exigences en qualité de services (QdS) de ces dernières ont un impact direct sur la consommation des ressources. Ainsi, une gestion efficace des ressources doit assurer non seulement les exigences QdS spécifiées dans le *SLA* pour éviter les surcoûts engendrés par la violation de ces dernières, mais aussi une consommation efficiente de ressource partagées. Par ailleurs, des changements inhabituels et soudains de la charge de trafic dus, par exemple, à des situations d'urgence (ex., mise à jour logiciel pour renforcer le niveau de sécurité des infrastructures) ou à des pannes, entraînent une dégradation critique des performances des services et des violations de *SLA* coûteuses. Dans ce contexte, les solutions qui se basent sur le surdimensionnement des infrastructures infonuagiques, des techniques de prédiction de la consommation de ressources en utilisant des seuils, ou des modèles mathématiques statiques (ex., régression linéaire basée sur des données extraites d'un historique hors-ligne) sont inefficaces. À la lumière de ce constat, nous proposons dans cette thèse une nouvelle solution intégrant des techniques d'apprentissage automatique pour gérer efficacement les ressources dans un environnement infonuagique. Notre solution se caractérise par: une prédiction dynamique et fiable de la consommation de ressources applicable à différents systèmes; un ajustement automatique et adaptif de la prédiction avec l'ajustement automatique des tailles optimales de la fenêtre coulissante des données d'apprentissage et celles à prédire; une détection automatique des variations anormales avec une mise en correspondance de ses métriques au niveau des services (ex., charge de trafic) et des ressources (ex., consommation de CPU/mémoire) et la détection de leur périodicité; une évaluation automatique de l'état du système (ex., variation normale due à l'adaptation des ressources ou bien variation anormale en raison d'une défaillance au niveau infrastructure matérielle ou logicielle) et la génération de notifications. Les algorithmes et les techniques proposés visent à optimiser le partage des ressources, à minimiser les coûts de leur consommation et à garantir les exigences du *SLA*. L'analyse des résultats obtenus à partir de différents tests expérimentaux a démontré la pertinence et l'efficacité de notre solution.

Mots-clés: gestion des ressources, prédiction, demande de ressources, apprentissage automatique, algorithme génétique, ajustement de la prédiction, divergence, mise en

correspondance, détection des valeurs extrêmes, périodicité, métrique de niveau ressource, métrique de niveau service, infonuagique.

Dynamic prediction of resource usage in virtualized environments

Souhila BENMAKRELOUF

ABSTRACT

Cloud computing environments are in a continual progress and offer a growing number of capacities to develop and to host new applications and services. In these environments, managing efficiently resources is considered as one of many and major challenges to face in order to respect negotiated service contracts (Service Level Agreement: SLA) required to implement these applications/services. In fact, the variability and unpredictability of traffic loads and Quality requirements Of Services (QoS) have a direct impact on the resource consumption. Thus, a performant resources management not only must guarantee the services quality requirements defined in the SLA contract to avoid the cost induced by their violation, but also an effective consumption of shared resources. Moreover, unusual and sudden changes in the traffic load, which are created by emergency cases (e.g.: software update to consolidate the infrastructures security) or breakdowns, produce a serious services performance degradation and costly SLA violations. In this context, the solutions that are based on the Cloud infrastructures resizing, resources consumption prediction methods using the thresholds or statistic models (e.g.: linear regression manipulating data retrieved from an offline historic) are inadequate. Considering this finding, we propose in our thesis a new solution that apply machine learning to manage efficiently resources in Cloud environments. Our solution is characterized by : a dynamic and reliable resources consumption prediction applicable to many systems ; automatic and adaptive tuning of the prediction combined with automatic adjustment of the sliding-window size of training data/predicted data (optimal size); an abnormal variations detection automatically and a mapping between metrics at service and resource levels plus a detection of their periodicity; automatic system state assessment (e.g. : a state update after a normal variation caused by resources adaptation or an abnormal change generated by a hardware/software failure in an infrastructure) and notification generation. Algorithms and technics that we propose have to optimize the resources sharing, to minimize consumption costs and to satisfy SLA requirements. The analysis of our experimental tests results show clearly the relevance and efficiency of our solutions .

Keywords: resource management, prediction, resource demand, machine learning, genetic algorithm, prediction adjustment, divergence, mapping, outlier detection, periodicity, resource level metric, service level metric, cloud computing.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ARIMA	Autoregressive Integrated Moving Average model
BPNN	Back Propagation Neural Network
CSCF	Call Session Control Functions
EM	Expectation Maximization Algorithm
ETSI	European Telecommunications Standards Institute
GA	Genetic Algorithm
QoS	Quality of Service
I-CSCF	Interrogating-Call Session Control Functions
MANO	Network Function Virtualization Management and Orchestration
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
P-CSCF	Proxy-Call Session Control Functions
S-CSCF	Serving-Call Session Control Functions
SLA	Service Level Agreement
SVR	Support Vector Regression
VIM	Virtualized Infrastructure Manager
VM	Virtual Machine
VNF	Virtualized Network Function

INTRODUCTION

0.1 Contexte

Ces deux dernières décennies, les technologies de l'information et de communication ont connu un essor fulgurant grâce notamment aux avancées dans les domaines de l'infonuagique et de la virtualisation. Ces deux derniers ont permis à plusieurs petites, moyennes et grandes entreprises, de migrer leurs infrastructures (ou systèmes) matérielles et/ou logicielles vers un environnement infonuagique qui est plus évolutive (échelonnable), plus tolérant aux pannes et offrant une grande flexibilité de gestion de services et des ressources. En effet, grâce à l'infonuagique, il est devenu possible de partager des ressources, des plateformes et des services entre plusieurs applications (ex. architecture multi-locataires). Par ailleurs, les technologies de virtualisation amènent quant à eux beaucoup d'avantages entre autres un partage de ressources optimal assurant une meilleure exploitation et des coûts faibles d'implantation favorisant son adoption auprès d'une large gamme d'entreprises. Néanmoins, garantir ces avantages ne peut être effectif sans une gestion optimale des ressources partagées qui est un défi majeur dans le contexte de la virtualisation et de l'infonuagique pour faire face à la demande grandissante entre autres en puissance de calcul et de stockage. En effet, dans de tels environnements virtualisés, distribués, partagés et dynamiques, il est nécessaire de gérer les ressources de manière automatique tout en garantissant les exigences de la qualité de service (QdS) spécifiées dans le contrat de niveau service (*Service Level Agreement* : *SLA*). Par ailleurs, une sous-estimation de la capacité des ressources allouées entrainerait une violation du *SLA,* alors qu'une surestimation de celle-ci engendrerait un surcoût dû à une inefficacité de la gestion de ces dernières. Par conséquent, il est important d'estimer la consommation des ressources d'une manière efficace et efficiente selon la demande effective afin d'assurer la qualité du service dans les environnements infonuagiques (Shen, Subbiah, Gu, & Wilkes, 2011). Une telle estimation repose principalement sur la capacité à prédire la charge de trafic transportée à travers l'infrastructure infonuagique et à anticiper l'évolution de la demande afin d'agir dans les délais requis.

0.2 Problématique

Dans des contextes aussi complexes et dynamiques que les systèmes et applications virtuels, la demande fluctuante en ressources doit être satisfaite en tout temps afin d'assurer la pérennité des systèmes et la qualité des services offerts aux utilisateurs. La pratique standard consiste à réaliser un surdimensionnement des infrastructures pour répondre aux exigences du SLA (Service Level Agreement) conduisant ainsi à une sous-utilisation des ressources et une surestimation des coûts de déploiement et de gestion de ces infrastructures par rapport aux besoins réels. Des solutions proposent des modèles de prédiction de la demande en ressources, mais leur fiabilité se trouve considérablement réduite en présence de changement dans la demande. En effet, ces modèles peuvent entraîner une dégradation notable des performances des services et des violations de SLA coûteuses face aux changements inhabituels et soudains de la charge de trafic, dus par exemple à des situations d'urgence (ex. mise à jour logicielle pour renforcer le niveau de sécurité) ou des défaillances des composants matériels ou logiciels de l'infrastructure.

Question de recherche

La question à laquelle cette thèse tente de répondre est: comment prédire de manière efficace, flexible et granulaire les ressources dans des environnements distribués et virtualisés et ce, d'une part en s'assurant de satisfaire les besoins en ressources et les exigences de QdS des différents applications/services et d'autre part, en optimisant l'utilisation des ressources afin de minimiser les coûts? Il s'agit donc de trouver une solution :

1. Assez générique (applicable à plusieurs systèmes virtualisés),
2. Proactive (capable de prédire le comportement du système et de notifier la planification des ressources à propos de son état et de sa future demande),
3. Adaptive (ajustement aux changements et fluctuations),
4. Dynamique (capable de répondre aux besoins en ressources des systèmes lors de leur évolution dans le temps).

0.3 Objectif Principal

Le principal objectif de cette thèse est de proposer une nouvelle solution automatique, proactive et adaptive qui permet une prédiction efficace et optimale des ressources fournies dans des environnements distribués et virtualisés et ce, en se conformant aux exigences du SLA et en minimisant les coûts reliés au déploiement et à la gestion de ces ressources. Basé sur des algorithmes d'apprentissage automatique, notre solution inclut principalement deux modules: un premier module qui assure une prédiction automatique de la consommation des ressources, et un deuxième module qui fournit une détection dynamique des comportements inhabituels dans l'utilisation des ressources.

La conception de notre solution et le choix des méthodes et des techniques employées ont été guidé par la volonté de proposer une approche portable en vue de fonctionner dans différents environnements et adaptive à divers comportements des systèmes. D'autre part, la généricité et le dynamisme de notre solution lui permettent d'évoluer dans le temps et par conséquent une plus grande longévité. Ces caractéristiques favorisent le déploiement de notre solution dans le cadre de micro-services, ainsi que dans des systèmes supportant des infrastructures à plus grande échelle.

La figure 0.1 illustre le composant proposé pour la prédiction des ressources lequel comporte les 2 modules cités ci-dessus. Ces derniers interagissent avec le module de planification de ressource qui n'est pas couvert dans le cadre de cette thèse. L'interaction entre les 3 modules est décrite dans le chapitre 2.

Par ailleurs, la figure 0.1 illustre un scénario de déploiement d'un tel composant. En effet, ce composant pourrait faire partie de l'infrastructure logicielle NFV MANO (*Network Functions Virtualization MAnagement and Network Orchestration*). Cette infrastructure a été développé par un groupe de travail portant le même nom au sein du groupe de spécification pour NFV (ISG NFV) de ETSI (*European Telcommunications Standards Institute*). Elle est dédiée à la gestion et à l'orchestration des ressources partagées dans un centre de données virtualisé (ETSI

GS, 2014). NVF MANO est subdivisé en trois blocs fonctionnels : Orchestrateur NFV (*NFV Orchestrator- NFVO*), le gestionnaire VNF (*Virtual Network Fucntion Manager – VNF manager*) et le gestionnaire des infrastructure virtualisées (*Virtualized Infrastructure Manager - VIM*). Parmi les fonctionnalités offertes par le VIM, figure la prédiction des ressources consommées. Ainsi, le composant proposé dans le cadre de cette thèse pourrait être intégré dans VIM.

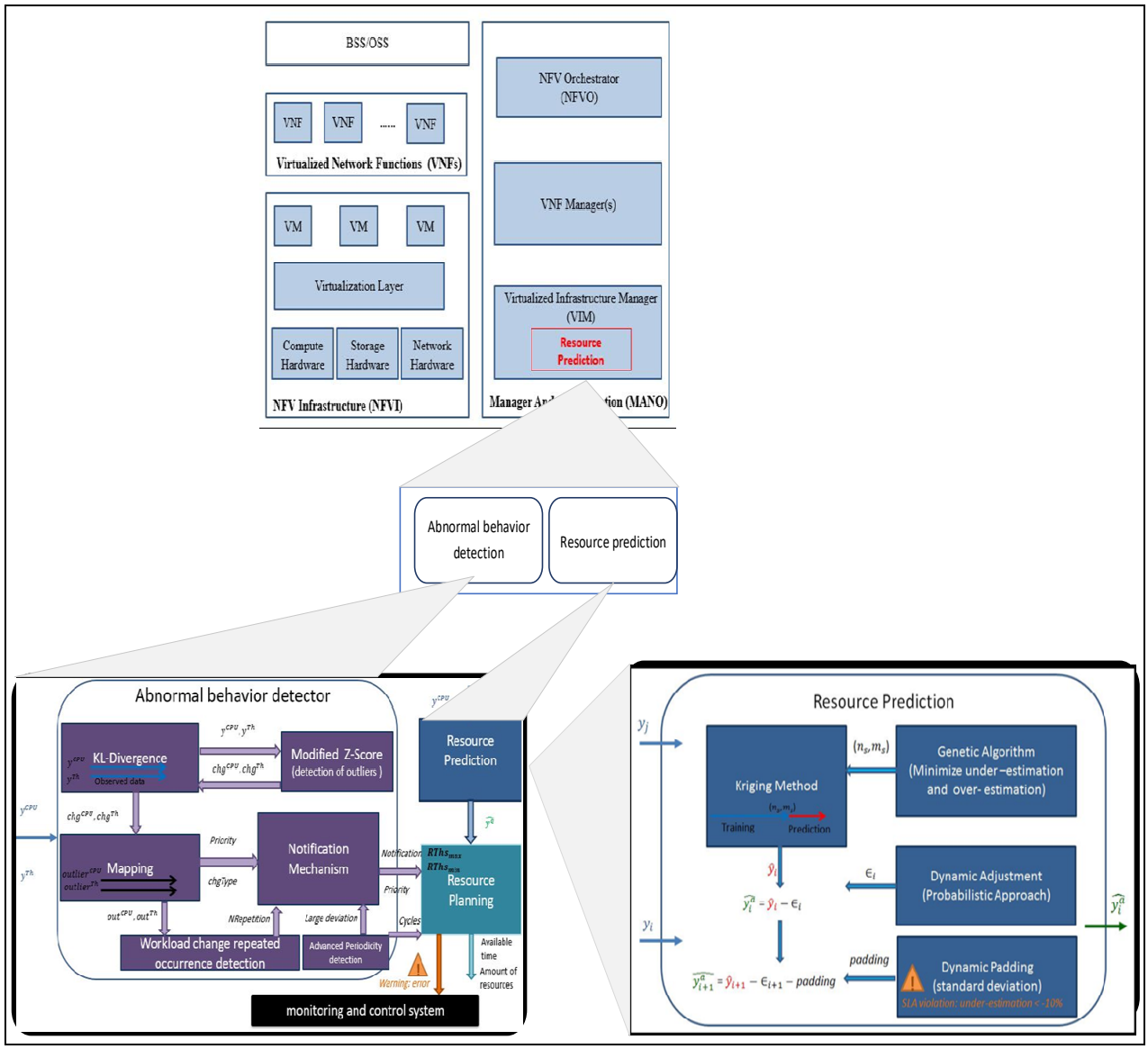

Figure 0.1 Aperçu des blocs fonctionnels du composant développé pour la prédiction automatique des ressources dans un environnement virtualisé

0.4 Méthodologie

Pour mettre l'accent sur les problèmes techniques soulevés, un environnement de virtualisation de systèmes a été déployé et différents profils de charge de trafic ont été configurés pour tester une architecture de services multimédias hébergée dans cet environnement. En effet, une architecture IMS (IP Multimedia Subsystem) a été virtualisée et distribuée dans cet environnement et de nombreux tests expérimentaux ont été effectués en imposant différentes charges de trafic à cette architecture afin de collecter différentes métriques au niveau ressources et au niveau service (ex., CPU et charge de trafic) pour chacun des composants CSCF (*Call Session Control Functions*) virtualisés de IMS à savoir P-CSCF (*Proxy-CSCF*), S-CSCF (*Serving-CSCF*), I-CSCF (*Interrogating-CSCF*) et HSS (*Home Subscriber Server*). La description détaillée de ce banc d'expérimentation est présentée dans le chapitre 2. Par ailleurs, plusieurs jeux de données (*datasets*) synthétiques ont été générés à partir des jeux de données collectés IMS pour garantir une variabilité dans les jeux de données testés. De plus, nous avons obtenu des jeux de données collectés à partir de systèmes distribués déployés dans des environnements de virtualisation réels. Tous ces jeux donnés nous ont permis de proposer de nouvelles approches de prédiction de la consommation de ressources afin d'assurer un partage des ressources efficace et fiable dans de tels environnements. Il est important de souligner que seule les métriques CPU et charge de trafic ont été utilisées pour le développement des solutions de prédiction, car la variabilité des autres métriques telles que la mémoire, la bande passante et l'espace disque collectées à partir du banc d'essai OpenIMS et celles recueillies à partir d'un déploiement réel de systèmes virtualisés n'était pas suffisante et concluante dans le cadre de cette recherche.

Le thème d'approvisionnement en ressources continue de faire l'objet de nombreuses études qui suggèrent une panoplie de solutions dont la fonction est d'estimer la quantité requise de ressources pour pouvoir anticiper les besoins des systèmes. En effet, la prédiction des demandes en ressources s'avère fondamentale pour la planification des besoins en temps réel et la gestion de ces ressources. Cependant, les solutions qui existent ne sont pas complètes. Certaines ne sont pas assez proactives et adaptatives dans la gestion des ressources. D'autres

sont spécifiques à un contexte particulier. Ainsi, ces solutions demeurent inefficaces pour des environnements, tels que l'infonuagique, caractérisés par une volumétrie très élevé en données et une charge de trafic dynamique et très variable.

Au début de ce travail de recherche, nous devions répondre à la question qui était : quel type de solution allons-nous choisir? Il ressortait de la série des tests expérimentaux que nous avons effectués, qu'une solution basée sur la modélisation, n'est fiable qu'en cas de stabilité du système en plus de sa spécificité. D'autre part, les solutions, qui utilisent l'apprentissage automatique, ont la capacité de s'adapter dynamiquement aux fluctuations sauf que leur fiabilité peut être affectée par la qualité et la variabilité des données collectées sur lesquelles se base l'apprentissage. Nous nous sommes inspirés de ce dernier type d'approches pour développer notre solution qui doit être : assez générique pour l'appliquer à un grand nombre de systèmes (jeux de données), dynamique et adaptive.

Dans le but d'améliorer la fiabilité du modèle déterminé par la méthode d'apprentissage machine, nous avons proposé une lecture dynamique des données en se basant sur une fenêtre coulissante. Le défi est de déterminer dynamiquement la taille optimale de cette fenêtre qui permet de minimiser les erreurs de prédiction et de répondre dans les délais requis pour ajuster les ressources en fonction de la demande. Par ailleurs, nos expérimentations ont démontré l'influence des facteurs taille des donnés d'apprentissage et taille des données prédites sur la fiabilité de la prédiction. À la lumière de ces constats, nous avons opté pour l'implantation d'une méthode d'apprentissage automatique afin de déterminer dynamiquement la taille des fenêtres d'apprentissage et de prédiction. En plus, cette méthode doit minimiser les erreurs de prédiction sans quoi nous avons des violations du SLA et une augmentation des coûts dues à des sous/surestimations de la demande en ressources. En fait, une prédiction automatique, adaptive et auto-ajustable (quand des erreurs se produisent) est en mesure de donner une estimation fiable de la demande future en ressources.

Néanmoins, la qualité de la prédiction se dégrade rapidement en présence de changements significatifs et imprévisibles dans le comportement du système. Ceci induit des périodes

relativement longues et coûteuses de violations du SLA en raison d'une sous-estimation ou bien un gaspillage des ressources due à une surestimation des demandes réelles. Des situations d'urgences ou des pannes générales sont des exemples de changements inhabituels et imprédictibles. Dans ce cas, comment pouvons-nous les détecter? Quelles sont les signes précurseurs de ces changements au niveau des services (ex., Charge de trafic) et au niveau des ressources (ex. consommation de CPU)? Comment allons-nous reconnaitre ces signes pour pouvoir les signaler et agir? Pour répondre à ces questionnements, notre démarche implique d'une part l'élaboration d'une détection automatique des changements inhabituels basée sur la combinaison et l'adaptation de techniques de détection de valeurs extrêmes (*outliers*), et d'autre part l'analyse du comportement du système en vue de trouver une périodicité ou des profils dans ce comportement.

Ainsi dans le premier article décrit au chapitre 2, nous proposons un nouvel algorithme capable de prédire d'une manière automatique la consommation des ressources. Pour y arriver, nous avons établi les objectives suivants :

- prédire automatiquement la demande en ressource grâce à l'implantation d'une méthode d'apprentissage automatique (prédiction fiable, adaptive et assez générique pour être applicable à différents systèmes);
- ajuster dynamiquement la prédiction en se basant sur l'estimation de la probabilité des erreurs de prédiction, et en suggérant une stratégie de padding pour réduire les sous-estimations (pouvant causer des violations du SLA) et les surestimations (pouvant engendrer des pertes de ressources);
- déterminer automatiquement les tailles optimales de la fenêtre de données d'apprentissage et de prédiction permettant de minimiser les erreurs de prédiction (surestimations et sous-estimations) en développant un algorithme d'optimisation basé sur l'algorithme génétique.

Dans le deuxième article décrit au chapitre 3, nous décrivons un nouvel algorithme pour détecter les changements inhabituels dans le comportement d'un système. Cet algorithme doit atteindre les objectifs suivants :

- détecter et évaluer automatiquement les changements anormaux dans un système au niveau des services et au niveau des ressources;
- analyser et identifier la périodicité ainsi que la présence des variations importantes dans le comportement d'un système;
- effectuer la mise en correspondance (Mapping) entre les métriques de niveau service et les métriques de niveau ressource;
- déterminer le type de changement survenu dans un système, sa répétition et son importance et par conséquent générer les notifications spécifiant l'action à appliquer et sa priorité pour la planification des ressources.

Dans le troisième article présenté au chapitre 4, nous décrivons une nouvelle approche pour estimer la charge de trafic et laquelle combine un filtre de Kalman avec une approche de régression à vecteurs de support (SVR-Support Vector Regression). Cette approche doit atteindre les objectifs suivants :

- estimer différents types de trafics ayant de fortes variations de leur charge qu'une infrastructure infonuagique pourrait gérer;
- comparer cette approche avec celle développée dans (St-Onge, Benmakrelouf, et al., 2020) qui combine le modèle Hull-White basé sur des équations différentielles stochastiques et l'algorithme génétique.

L'analyse présentée dans ce chapitre a permis de développer une nouvelle approche statique de détection de périodicité dans les charges de trafic ayant différentes amplitudes, formes et périodicité (St-Onge, Kara, Abdel Wahab, Edstrom, & Lemieux, 2020). Cette même approche, nous l'avons adaptée pour une détection automatique de cette périodicité et laquelle est décrite dans le chapitre 3.

0.5 Contributions

Les principales contributions de cette thèse peuvent être résumées comme suit:

- un nouvel algorithme pour une prédiction automatique et adaptive de la demande en ressources dans les systèmes virtualisés sans aucune connaissance ou hypothèse préalable sur leurs comportements internes.
- la détermination dynamique des tailles optimales de la fenêtre coulissante et des données d'apprentissage et des données prédites.
- un nouvel algorithme pour une détection dynamique des variations anormales, une mise en correspondance des métriques du niveau de services avec celles du niveau de ressources dans des systèmes virtualisés, et une détection de la périodicité.
- une évaluation efficace et automatique de l'état du système et la génération de notifications pour répondre aux besoins en ressources ou lancer des alertes.

0.6 Publications et brevet

- Souhila Benmakrelouf, Nadjia Kara, Hanine Tout, Rafi Rabipour and Claes Edstrom. (2019). Resource needs prediction in virtualized systems: generic proactive and self-adaptation solution. Elsevier Journal of Network and Computer Applications (IF-5.273), 148, 1-16. (Souhila Benmakrelouf, Kara, Tout, Rabipour, & Edstrom, 2019).

- Souhila Benmakrelouf, Cédric St-Onge, Nadjia Kara, Hanine Tout, Claes Edstrom and Yves Lemieux. (2020). Abnormal behavior detection using resource level to service level metrics mapping in virtualized systems. Future Generation Computer Systems (IF-5.768), 102, 680-700. (Souhila Benmakrelouf et al., 2020).

- Cédric St-Onge, Souhila Benmakrelouf, Nadjia Kara, Hanine Tout, Claes Edstrom and Rafi Rabipour. (2020) Advanced workload modelling for cloud systems. Accepted for publication in Journal of Cloud Computing (IF-2.960).

- Souhila Benmakrelouf and Nadjia Kara. Resource Needs Prediction in Virtualized Systems: Generic Proactive and Self-Adaptive Solution. Regular Patent reference number P73928 WIPO, January 2019.(Souhila Benmakrelouf & Kara, 2019)

0.7 L’organisation de la thèse

Nous présentons notre travail sous forme de thèse par articles. Ainsi, nous avons commencé par une revue générale de littérature suivie par les chapitres qui détaillent chacun de nos articles. Finalement, nous concluons la thèse et nous proposons des travaux futurs.

CHAPITRE 1

REVUE DE LITTÉRATURE

Nous introduisons dans ce chapitre quelques définitions concernant les méthodes d'apprentissage automatique. Nous présentons également différentes approches proposés dans des travaux de recherches récents reliés au sujet de cette thèse. Cette revue de littérature permettra ainsi de positionner ce travail de recherche dans la littérature tout en soulignant nos contributions comparativement à celles passées en revue. Toutefois, une analyse détaillée des travaux de recherches est effectuée dans chacun des chapitres présentant les solutions proposées.

1.1 Définitions

Nous définissons dans cette section l'apprentissage automatique et nous introduisons brièvement les méthodes appliquées dans notre approche pour la prédiction de la consommation des ressources dans des environnements virtualisées. Cependant, ces méthodes sont expliquées en détail dans les chapitres suivants.

1.1.1 Apprentissage automatique

L'apprentissage automatique (Boutaba et al., 2018) permet à un système d'analyser les données et déduire des connaissances. Les techniques d'apprentissage automatique sont conçues pour identifier et exploiter des modèles cachés dans les données pour divers problèmes, tels que les problèmes de regroupement de données (*clustering*), la prédiction, la classification et la régression.

L'apprentissage automatique (Boutaba et al., 2018) nécessite des données représentatives pour construire un modèle efficace pour un problème donné. La collecte de données est une étape importante, car elle permet de garantir la qualité et la taille des données sur lesquelles reposent le processus d'apprentissage. Elle peut être configurée en mode hors ligne ou en mode en ligne.

1.1.2 Méthode Kriging

La méthode Kriging (Gratton, 2002) a été introduit la première fois par l'ingénieur minier sud-africain D.G. Krige pour déterminer la distribution spatiale de minerais à partir d'un ensemble de forages. Basé sur l'interpolation spatiale, la méthode Kriging suppose une corrélation spatiale entre les données observées. Elle utilise les méthodes statistiques pour la prédiction. Ainsi, pour prédire la valeur $\widehat{y_p}$, la méthode Kriging calcule la combinaison linéaire pondérée des données observées y_i dans le voisinage (Van Beers & Kleijnen, 2004)

$$\widehat{y_p} = \sum_{i=1}^{n} \omega_i \, y_i \tag{1.1}$$

où ω_i est le poids associé à y_i , et $\sum_{i=1}^{n} \omega_i = 1$.

Pour réaliser la prédiction avec la méthode Kriging, il faut trouver les règles de dépendance entre les valeurs observées et créer le semivariogramme empirique qui fournit des informations à propos de l'autocorrélation spatiale des données. L'ajustement d'un modèle (ex. linéaire , exponentielle, sphérique) au semivariogramme empirique permet de déduire les poids ω_i et évaluer les données interpolées. Les poids ω_i sont calculés en résolvant le système d'équations linéaires suivant (Gratton, 2002) (G. Liu, Xu, & Chen, 2014) :

$$\begin{bmatrix} A & 1 \\ 1^T & 0 \end{bmatrix} \begin{bmatrix} \vec{\omega} \\ \mu \end{bmatrix} = \begin{bmatrix} B \\ 1 \end{bmatrix} \tag{1.2}$$

où

A_{ij} est la valeur du semivariogramme correspondant à la distance h_{ij} entre y_i et y_j, B_{ip} est la valeur du semivariogramme calculé selon la distance h_{ip} entre y_i et y_p(valeur à prédire), $\vec{\omega}$ est le vecteur des poids et μ est le multiplicateur de Lagrange.

La méthode Kriging permet l'adaptabilité aux divers comportements des systèmes (linéaire, non-linéaire, multimodal) avec une complexité qui varie avec la taille des données traitées. Toutefois, sa performance diminue si le modèle du semivariogramme est erroné ou la taille des données est insuffisante.

1.1.3 Méthode Vecteur de support pour la régression

La méthode machine à vecteur de support (*Support Vector Machine : SVM*) est utilisée principalement pour la classification des données. Elle permet de séparer les données en deux classes de données d'apprentissage à l'aide d'hyperplan linéaire. Si les classes ne peuvent pas être séparées linéairement, les données sont associées à un espace de caractéristiques à haute dimension en utilisant une fonction de mise en correspondance (*mapping*) non-linéaire appelée fonction noyau (ex: linéaire, polynomiale, fonction de base radiale). Il s'agit d'identifier l'hyperplan optimal maximisant la marge entre les deux classes et de déterminer les paramètres optimaux w et b (Cortes & Vapnik, 1995) (Smola & Schölkopf, 2004):

$$f(x) = wK(x) + b \tag{1.3}$$

où

x: données d'entrée, w: vecteur de pondération, b: paramètre de biais et $K(x)$: fonction noyau

La méthode vecteur de support pour la régression est une variante de SVM. Elle effectue une régression, prédisant des variables ordonnées et continues. Les deux méthodes utilisent des algorithmes très similaires, mais prédisent différents types de variables. Toutefois, le choix de la fonction noyau, la taille des données et le temps d'exécution lors des phases d'apprentissage et de tests sont parmi les limitations de la méthode SVR.

1.1.4 Algorithme génétique

Les algorithmes génétique (AGs) (Mitsuo & Runwei, 2000) (Tout, Talhi, Kara, & Mourad, 2019) sont une abstraction de l'évolution biologique. Ils s'inspirent du processus d'évolution naturelle pour résoudre un problème d'optimisation. Les AGs sont des méthodes heuristiques

qui visent à trouver la solution optimale en simulant la propagation des individus les plus aptes sur des générations consécutives. L'adéquation d'une solution (*fitness*) est basée sur une fonction qui vise à minimiser ou maximiser un ou plusieurs objectifs. Grâce à des opérateurs de croisement, de mutation et de sélection, l'algorithme génétique est capable de générer des individus diversifiés et de trouver les meilleurs d'entre eux.

1.2 Prédiction des ressources

La prédiction de la demande en ressources dans des environnements virtualisés et distribués tel que l'infonuagique rencontre plusieurs difficultés à cause de la complexité du comportement des applications et leurs interactions avec leur environnement lequel est dynamique, l'hétérogénéité et la distribution des composants constituant ces applications. (Hu, Jiang, Liu, & Wang, 2013) Dans un contexte aussi complexe et dynamique, des approches basées sur des métaheuristiques ou la modélisation des comportements des applications sont insuffisantes. Par ailleurs, le dynamisme et l'incertitude qui caractérisent l'environnement de l'infonuagique compliquent l'estimation de la consommation des ressources et impliquent des sous/sur-approvisionnements et des dégradations de la QdS induisant ainsi des violations du SLA (Vashistha & Verma, 2020). D'où le besoin de développer des approches automatiques et proactives.

Grâce à une prédiction fiable et automatique, le sur-approvisionnement ainsi que le sous-approvisionnement des ressources sont minimisés voire évités, ce qui permet une optimisation des coûts de déploiement et de gestion des infrastructures de virtualisation tout en garantissant la qualité du service. Ainsi, dans de récentes études, les chercheurs ont proposé de prédire la demande en ressources à travers l'apprentissage automatique basé par exemple sur : une approche combinant un ensemble de modèles de régression (ex. SVM-Support Vector Machine, réseau de neurones, régression linéaire, etc.) pour la prédiction de l'utilisation du CPU (Kaur, Bala, & Chana, 2019), le modèle autorégressif à moyennes mobiles intégrées (*Autoregressive Integrated Moving Average model :ARIMA*) et la moyenne mobile exponentielle pour prédire la consommation du CPU et dimensionner les VMs (da Rosa Righi,

Correa, Gomes, & da Costa, 2020), l'apprentissage profond (*Long Short-Term Memory :* LSTM) (Chien, Lai, & Chao, 2019) (Gao, Wang, & Shen, 2020), et l'apprentissage par renforcement basé sur l'algorithme Q-learning afin de planifier les ressources requises (VMs) (X. Chen et al., 2020), etc.

Dans (Hsieh, Liu, Buyya, & Zomaya, 2020), les auteurs ont présenté une approche de consolidation des machines virtuelles (VM) permettant de réduire la consommation d'énergie tout en maintenant la QdS. Pour atteindre ces objectifs, ils ont considéré l'utilisation des ressources courante et future. La consommation future de ressources est déterminée grâce au modèle de prédiction de Gray-Markov. Les résultats de leur expérimentation ont démontré une réduction du nombre de migrations de VM et de la consommation de l'énergie.

Dans (J. Chen & Wang, 2020), les auteurs ont proposé de traiter le problème de la prédiction de la consommation de ressources dans un environnement infonuagique en appliquant un algorithme adaptatif de prédiction à court terme basé sur le modèle ARIMA. Cet algorithme utilise en premier lieu la méthode d'analyse en composantes principales (ACP) et la détection des valeurs extrêmes afin de prétraiter les données. Par la suite, la méthode de prédiction ayant la meilleure précision est sélectionnée parmi celles proposées pour anticiper la demande en ressources. Les auteurs ont également considéré l'ajustement des erreurs de prédiction. L'évaluation de leur approche a montré une amélioration de la précision comparativement à d'autres d'algorithmes de prédiction tels que ARIMA et le réseau de neurones de rétropropagation (Back Propagation Neural Network : BPNN).

Également, d'autres approches pour la prédiction de l'utilisation des ressources (ex. trafic du réseau, utilisation du CPU) sont présentées dans de nombreuses études. Ces approches sont basées sur des modèles tels que : la régression linéaire multiple (Lloyd et al., 2013), SVR et filtre de Kalman (Hu et al., 2013), la méthode Kriging (Gambi, Pezze, & Toffetti, 2016) et les modèles autorégressifs à moyenne mobile (Autoregressive–Moving Average : ARMA) (Hoong, Tan, & Keong, 2012) (Iqbal & John, 2012). Les résultats de ces approches sont prometteurs, mais ils restent relatifs aux contextes de leur évaluation et ne peuvent être

généralisés. Ces approches n'incluent pas un ajustement de la prédiction et leur performance est affectée en présence de fluctuations et de pics dans la charge de travail par manque d'adaptabilité.

Discussion:

Basées sur des méthodes d'apprentissage automatique et théories (ex., Markov and Gray), les approches proposées pour la prédiction de la demande de ressources ont permis d'améliorer les performances tout en réduisant les coûts de déploiement et de gestion de ressources et la consommation d'énergie. Toutefois, la complexité des algorithmes utilisés, en termes de temps d'exécution et le manque de généricité (applicable à des jeux de données variés) des solutions proposées sont tous des aspects qui restent à améliorer. Ainsi comparativement à ces travaux de recherche et ceux listés dans les chapitres 2,3 et 4 et aux meilleurs de nos connaissances, notre solution s'inscrit dans cette perspective de recherche, elle se distingue par l'exploitation de méthodes d'apprentissage automatique permettant d'offrir une solution automatique, adaptive et assez générique pour être appliquée à différents jeux de données. Notre approche permet également un ajustement automatique et adaptif des erreurs de prédiction ce qui permet une diminution significative des sous/surestimations et par conséquent une réduction des violations du SLA et de la perte de ressources.

1.3 Détection des changements anormaux dans le comportement des systèmes virtualisés

Pour garantir la QdS, les fournisseurs d'infrastructures infonuagiques gèrent les ressources de sorte que les exigences négociées dans le contrat de service soient respectées. Or, dans des situations critiques tels que des mises à jours logicielles urgentes, des défaillances matérielles ou logicielles, ou les outils de monitorage défaillants en termes de détection des changements anormaux et de la mise en correspondance (*Mapping*) de ces changements entre le niveau de ressources et celui des services, cette QdS est affectée entrainant des coûts supplémentaires dus à des violations prolongées du SLA ou bien une sous-utilisation des ressources. Plusieurs chercheurs se sont donc intéressés à cette problématique vu son importance et son impact sur

les applications et les fournisseurs d'infrastructures infonuagiques. Les approches proposées dans de récentes études portent sur différentes questions de cette problématique tels que la détection des comportements anormaux (Toumi, Brahmi, & Gammoudi, 2020) (Areeg & Pahl, 2019), les pannes matérielles et logicielles (Gao et al., 2020), la mise en correspondance entre les métriques au niveau de ressources et les paramètres de la QdS (ex. charge de trafic) en utilisant les réseaux de neurones artificiels (Stavroulas et al., 2019), ou les intrusions et attaques malveillantes (Yadav, 2019) (Aldribi, Traoré, Moa, & Nwamuo, 2020).

Afin de prédire la charge de travail et de détecter en temps réel les changements, Toumi et son équipe (Toumi et al., 2020) ont utilisé une technique de forage de flux de données (*stream mining*), à savoir, *Hoeffding Adaptive Tree* qu'ils ont associée à un ensemble de détecteur de dérives (*drift detectors*). L'évaluation a montré une bonne précision de la solution proposée face aux changements dans la répartition de la charge avec un faible temps de réponse et un faible usage de la mémoire.

D'autre part, une approche de prédiction des échecs des tâches dans l'infonuagique résultant des défaillances matérielles ou logicielles, est proposée par Gao et *al.* dans (Gao et al., 2020) en utilisant l'apprentissage profond (*multi-layer Bidirectional Long Short Term Memory*). Les résultats de leur expérimentation effectuée sur des données extraites de *Google Cluster Trace* démontrent une précision de la détection des défaillances de 93%.

Dans l'étude de Stavroulas et *al.* (Stavroulas et al., 2019), une approche basée sur l'apprentissage automatique, utilisant des réseaux de neurones artificiels et l'algorithme génétique, a permis de mettre en correspondance (*Mapping*) des métriques au niveau des ressources (ex. consommation de la mémoire) et des paramètres de la QoS (ex. charge de trafic) pour déterminer les demandes en ressources spécifiques aux applications déployées.

Dans le but de déceler des anomalies dans le trafic des réseaux (Salem, Nait-Abdesselam, & Mehaoua, 2012) (Li & Wang, 2012), des points extrêmes et des points de changement (Takeuchi & Yamanishi, 2006), ou des pannes (Youssef, Delpha, & Diallo, 2016), différentes

études ont employés des techniques de détection de la divergence des données ainsi que des techniques pour la détection des points extrêmes. Parmi ces techniques, on retrouve, par exemple, la divergence de Kullback-Leibler, la divergence de Jensen-Shannon, et les scores à moyenne mobile appliquées sur des données historiques (hors ligne - *offline*) ou collectées à l'aide des outils de monitorage (temps réel - *real-time*). Dans la plupart des solutions proposées, la détection de la périodicité et des modèles ainsi que la mise en correspondances des métriques au niveau des ressources avec celles au niveau services ne sont pas considérées.

Discussion :

La détection des changements significatifs et de fortes fluctuations dans le comportement des applications (ex. changement de la charge de trafic) et de la consommation de ressources (ex. changement de la consommation en CPU) est un important défi dans la gestion des ressources dans un environnement infonuagique. Afin d'éviter des coûts supplémentaires induits par de telles situations, différentes approches sont présentées. Les plus récentes de ces approches se basent essentiellement sur les méthodes d'intelligence artificielle. Bien que ces études aient permis la détection de multiples types de changements, la précision et la complexité des méthodes utilisées restent à améliorer. Ainsi, notre solution s'inscrit dans cette optique de recherche. Elle se distingue par la détection des changements anormaux, l'identification de la périodicité et de profils dans ces changements, ainsi que leurs mise en correspondance aux niveaux services et ressources permettant ainsi une évaluation plus précise de la consommation de ressources afin d'assurer une prise de décision la plus adéquate.

1.4 Conclusion

Dans ce chapitre, nous avons passé en revue les approches les plus récentes et pertinentes à la prédiction automatique et adaptive de la consommation de ressources dans un environnement infonuagique. Tel que discuté précédemment, les approches proposées présentent certaines limitations qui doivent être résolues afin d'assurer une prédiction de ressources qui satisfait le contrat de services négocié avec chaque utilisateur et réduire les coûts induits par une sous-utilisation de ces dernières. Chacun des chapitres qui suivent permet de résoudre une des

limitations identifiées et décrit une partie de la solution complète proposée dans le cadre de cette thèse.

CHAPITRE 2

RESOURCE NEEDS PREDICTION IN VIRTUALIZED SYSTEMS: GENERIC PROACTIVE AND SELF-ADAPTIVE SOLUTION

Souhila Benmakrelouf [a], Nadjia Kara [b], Hanine Tout [c], Rafi Rabipour [d], Claes Edstrom [e]

[a, b, c] Department of software engineering and IT, École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[d, e] Ericsson Canada,
8275 route Transcanadienne, Saint-Laurent, Québec, Canada, H4S 0B6

Published in "Journal of Network and Computer Applications",
September 2019

2.1 Abstract

Resource management of virtualized systems in cloud data centers is a critical and challenging task due to the fluctuating workloads and complex applications in such environments. Over-provisioning is a common practice to meet service level agreement requirements, but this leads to under-utilization of resources and energy waste. Thus, provisioning virtualized systems with resources according to their workload demands is essential. Existing solutions fail to provide a complete solution in this regard, as some of them lack proactivity and dynamism in estimating resources, while others are environment- or application-specific, which limits their accuracy in the case of bursty workloads. Effective resource management requires dynamic and accurate prediction. This work presents a novel prediction algorithm, which is generic, and can thus be applied to any virtualized system, is able to provide proactive estimation of resource requirements through machine learning techniques, and is capable of real-time adaptation with padding and prediction adjustments based on prediction error probabilities in order to reduce under- and over-provisioning of resources. In several virtualized systems, and under different workload profiles, the experimental results show that our proposition is able to reduce under-estimation by an average of 86% over non-adjusted prediction, and to decrease over-estimation by an average of 67% versus threshold-based provisioning.

Keywords: prediction, resource needs, machine learning, kriging, genetic algorithm, adjustment.

2.2 Introduction

Virtualization is one of the key technologies leveraged to provide scalability, better management flexibility, optimized resource sharing and lower costs in data centers. To capitalize on this technology, it is essential to provision virtualized systems with resources dynamically according to their workload demands. However, the complexity of virtualized systems and applications, their fluctuating resource demands over time (Shyam & Manvi, 2016), and their dynamic and heterogeneous environments all impose a real challenge in resource management, which requires optimizing resource utilization while avoiding Service Level Agreement (SLA) violations. A common practice is to over-provision resources to meet various SLA requirements established with clients. However, this increases the costs incurred in data centers in terms of energy consumption (Goudarzi & Pedram, 2016) and capital expenditure since more resources have to be available. Scalable and elastic allocations of resources is necessary and crucial for the dynamic adjustment of resource capacity to actual demand in real time, while minimizing SLA violations and delays in resource scaling.

Effective and accurate prediction of resource demands is fundamental to real-time resource needs planning and virtualized resource management in data centers. It helps to meet service-level agreement stipulations (by minimizing under-provisioning), anticipate needs in terms of middleboxes (e.g., Load Balancer, Firewall) and lay the groundwork for proactive job scheduling. Thus, consequently this will improve the resource usage, service performance and reduce costs (by minimizing over-provisioning) (Shyam & Manvi, 2016). Several studies have proposed diverse techniques to address these issues [(Shyam & Manvi, 2016), (Hoong et al., 2012), (Iqbal & John, 2012), (Lloyd et al., 2013), (Liang, Cao, Wang, & Xu, 2011), (Hu et al., 2013), (Gambi et al., 2016)], however none of them has provided a complete solution. Some of these approaches do not offer proactive and adaptive management of resources or even consider SLA requirements. Moreover, some of these solutions are environment-specific or

application-specific. This limits their accuracy in the case of unexpected and large amounts of data (Shyam & Manvi, 2016), which is a major drawback in the cloud context, where we are dealing with highly dynamic and bursty workloads.

To address these limitations, we propose in this work a novel solution, which is generic enough to be applied to any virtualized system or application. It is able to dynamically generate and adjust predictions in real time and offers proactivity in terms of estimating resource demand by anticipating future changes in the system. Our approach provides an algorithm for the dynamic, accurate and effective prediction of resource needs by developing and leveraging different methods and techniques. Black-box prediction methods derive models from the system behavior without requiring any knowledge of the system internals (Gambi, 2012). The adaptability and the efficiency of these methods make them appropriate for virtualized, dynamic and complex environments such as data centers. The proposed algorithm also employs machine learning methods and time series to remain a few steps ahead in the dynamic estimation of resource needs. Furthermore, because prediction is not always sufficiently accurate, adjustments are sometimes needed. Therefore, a dynamic adjustment technique is devised and employed in the prediction algorithm to reduce the under- and over-estimation of resources. A thorough experimentation was also conducted to study the efficiency and performance of the proposed algorithm with different systems and workloads.

European Telecommunications Standards Institute (ETSI) introduced Network Function Virtualization (NFV) concept to reduce cost and accelerate service deployment. The high-level NFV framework proposed by ETSI identified three main working domains (see Figure 2.1): Virtualized Network Function (VNF), NFV Infrastructure (NFVI) and NFV Management and Orchestration (MANO). The latter covers the orchestration and lifecycle management of physical and/or software resources and the lifecycle management of VNFs. The proposed resource prediction Algorithm can be integrate into MANO and more specifically as a part of Virtualized Infrastructure Manager (VIM).

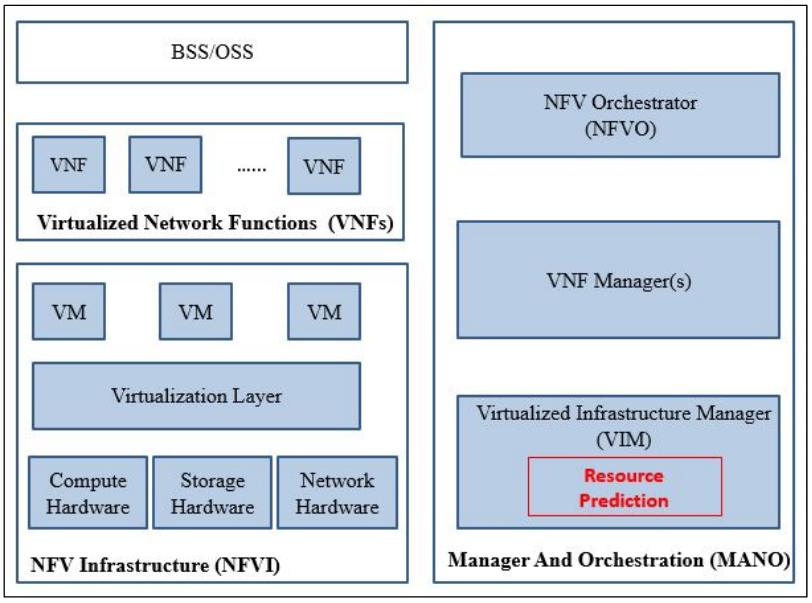

Figure 2.1 High-level NFV framework (ETSI, 2013)

The main contributions of this work are threefold:

1. Novel algorithm for dynamic and multi-step ahead prediction of resource needs in virtualized systems without any prior knowledge or assumptions on their internal behaviors,
2. Dynamic and adaptive adjustment of prediction based on the estimated probability of prediction errors, and padding strategy to reduce under-estimation (SLA violations) and over-estimation (resource loss) of resource needs,
3. Dynamic determination of the optimal sizes of the sliding window and the predicted data that minimize under- and over-estimations through Genetic Algorithm (GA) intelligence.

The rest of the paper is organized as follows. First, we review the state of the art related to resource demand prediction in the context of virtualized system resource management. Second, we present our proposed approach and we explain the algorithm, the methods and the strategies we used. Third, we evaluate the performance of the prediction algorithm. Finally, we analyze and discuss the main results and conclude the article.

2.3 Related work

Focus on resource management of virtualized systems as a research area has recently been gaining steam, and several techniques have been proposed in this regard. In this section, we study existing work on the techniques used in this domain. To highlight our contributions, in Table 2.1, we classify the most recent approaches and compare them based on key features needed for efficient prediction of resources in virtualized systems.

2.3.1 Time series

Time series (Fu, 2011) is a collection of observations presented chronologically, characterized by a large data size, high dimensionality and continuous update. In his review, Fu (Fu, 2011) categorized time series data into representation and indexing, similarity measure, segmentation, visualization and mining. He also considered the similarity measure and segmentation as the core tasks for various time series mining tasks. To analyze time series data, various methods and techniques have been used, namely, Support Vector Regression, auto-regression, Expectation Maximization Algorithm, hidden Markov models, and Fourier transforms. When it comes to the segmentation of time series into subsequences for preprocessing or trend analysis, an important observation has proved the effectiveness of a dynamic approach that uses a variable window size, rather than a fixed one, to flexibly identify the time points.

In the same context, the studies conducted in (Islam, Keung, Lee, & Liu, 2012) (Tran, Tran, Nguyen, & Le, 2016) revealed that the input window size impacted the prediction model accuracy, which has been improved by the use of the sliding window strategy. Contrary to all historical data, the sliding window data allows prediction models to follow the trend of recently observed data and the underlying pattern within the neighborhood of predicted data mainly in the case of highly fluctuant and bursty workloads. Therefore, it allows achieving more accurate predictions.

2.3.2 Prediction

Prediction approaches can be mainly categorized into two classes. The first category is based on models deduced from the system behavior analysis. Existing approach derived from such analytical models focus mainly on auto-regression and moving averages (Hoong et al., 2012) (Yanhua, Meina, Zhijun, & Junde, 2011) (Iqbal & John, 2012), multiple linear regression (Lloyd et al., 2013), Fourier transform and tendency-based methods (Liang et al., 2011) (Gandhi, Yuan, Gmach, Arlitt, & Marwah, 2011), and cumulative distribution functions (Goudarzi & Pedram, 2016). Researchers in (Lloyd et al., 2013) evaluated the relationships between resource utilization (CPU, disk and network) and performance using multiple linear regression technique to develop a model that predicts application deployment performance. Their model accounted for 84% of the variance (coefficient of determination) in predicting the performance of component deployments. However, all these models are static and non-adaptive to unexpected changes in system behavior or environment. This is due to their use of configuration-specific variables. For its part, the second category of resource prediction approaches is based on machine learning methods. Applications in this class are dynamic and adaptive, but less accurate than the model-based approaches as they may be affected by the non-reliability of measurement tools, which may lead to erroneous values.

To achieve predictions that are both dynamic and more accurate, recent research studies have proposed combining both approaches into hybrid solutions (Gambi, 2012) (Amiri & Mohammad-Khanli, 2017). Numerous studies have proposed machine learning methods for the dynamic resource usage prediction. These methods include the Kalman filter (Zhang-Jian, Lee, & Hwang, 2014) (W. Wang et al., 2012), Support Vector Regression (SVR) (Hu et al., 2013) (Huang et al., 2013) (Z. Wei, Tao, ZhuoShu, & Zio, 2013), Artificial Neural Networks (ANN) (Islam et al., 2012) (Tran et al., 2016) (Ma et al., 2017), and Bayesian models (Shyam & Manvi, 2016). The authors in (Shyam & Manvi, 2016) proposed a Bayesian model to determine short- and long-term virtual resource requirements for applications based on workload patterns at several data centers, during multiple time intervals. The proposed model (Shyam & Manvi, 2016) was compared with other existing work based on linear regression

and support vector regression, and the results showed a better performance for the Bayesian model in terms of mean squared error. Nevertheless, as the proposed model was based on workload patterns generated from resource usage information during weekdays and weekends, it may be unable to respond to quick and unexpected changes in resource demands.

Researchers in (Hu et al., 2013) suggested a multi-step-ahead CPU load prediction method based on SVR and an integrated Smooth Kalman Filter to further reduce the prediction error (KSSVR). The results of their experiments showed that KSSVR had the best prediction accuracy, followed successively by standard SVR, Back-Propagation Neural Network (BPNN), and the Autoregressive model (AR). Nevertheless, when dealing with small and fixed size training data from heavily loaded and highly variable interactive machines, KSSVR provides decreased prediction accuracy.

Gambi et al. (Gambi et al., 2016) proposed self-adaptive cloud controllers, which are schedulers that allocate resources to applications running in the cloud, based on Kriging models, in order to meet the quality of service requirements while optimizing execution costs. They used Kriging models to approximate the complex and a-priori unknown relationships between: the non-functional system properties collected with runtime monitors (e.g., availability, and throughput), the system configuration (e.g., number of virtual machines), and the service environmental conditions (e.g., workload intensity, interferences). Their test results confirmed that Kriging outperforms multidimensional linear regression, multivariate adaptive regression splines and Queuing models. However, the relatively poor performance of controllers using pure Kriging models revealed that Kriging-based controllers require the availability of a larger set of training values for improved performance.

2.3.3 Adjustment and padding

To avoid under-estimation of resource needs, a prediction adjustment has been proposed in several studies. The adjustment was introduced as a padding to be added to the predicted data as a prediction cap (Qazi, Li, & Sohn, 2014). This padding was prefixed (e.g., 5%) (Qazi et al.,

2014) or calculated dynamically using various strategies. The latter include measuring the relationship between the padding value and the confidence interval, which is defined as the probability that real demand is less than the cap (Jing, Jie, Guangquan, & Guodong, 2013), or considering the maximum of the recent burstiness of application resource consumption using fast Fourier transform and recent prediction errors through a weighted moving average (Shen, Subbiah, Gu, & Wilkes, 2011), or using the confidence interval based on the estimated standard deviation for prediction errors (J. Liu, Shen, & Chen, 2016).

2.3.4 Proposed approach

The time and effort needed to build analytical models (off-line modeling) limit their usefulness in dynamic and real-time applications despite their accuracy (Z. Wei et al., 2013). Based on historical observed data, it is known that these models are not able to capture behavioral changes in applications or the systems. Furthermore, techniques based on threshold rules that assume linearity and stability in the system behavior are not realistic solutions, given the complexity and the unpredictable behavior of current systems, as well as their internal and external interactions (Gambi et al., 2016). Furthermore, being application-specific, these solutions lack the ability to adapt to cloud dynamics, where their models are generated based on the analysis of a specific application or system for a given environment and behavior. In contrast, data-driven approaches relying on machine learning methods are able to outperform analytical models and adapt to changes by deriving models from the system behavior without requiring any knowledge of the system internals. However, existing resource prediction models in the cloud consider an excessive allocation of resources in order to avoid SLA violations during peak demand periods (Shyam & Manvi, 2016). This leads to a waste of resources and energy, and increases operating costs. Table 2.1 provides the taxonomy of the most recent relevant approaches in these contexts.

Our approach differs in many aspects from the studies in the literature, and provides a generic, dynamic, and self-adaptive solution for resource prediction. In this proposition, we leverage black-box techniques to provide a generic solution, which can be applied to any system, with

Table 2.1 Taxonomy of prediction approaches

<table>
<tr><th rowspan="2">Study</th><th colspan="8">Criteria</th></tr>
<tr><th>Target</th><th>Granularity</th><th>Model</th><th>Adaptability</th><th>Prediction based on</th><th>Adjustment / Padding</th><th>SLA Sensitive</th><th>Energy Aware</th></tr>
<tr><td>(Shyam & Manvi, 2016)</td><td>Short- and Long-term Prediction of Resource Demand</td><td>Applications in Cloud</td><td>Bayesian Model</td><td rowspan="6">Non Adaptive</td><td>Cyclical and / or Seasonal patterns</td><td rowspan="7">Does Not Apply</td><td rowspan="7">Does Not Apply</td><td>Does Not Apply</td></tr>
<tr><td>(Hoong et al., 2012)</td><td>Short-term Prediction</td><td rowspan="2">Network Traffic</td><td>ARMA time series</td><td rowspan="2">Historical Data, Training Data</td><td rowspan="2">Performance and Energy Overhead of Predictors</td></tr>
<tr><td>(Iqbal & John, 2012)</td><td>One-step ahead Prediction</td><td>Double Exponential Smoothing, ARMA, Artificial Neural Network and Wavelet</td></tr>
<tr><td>(Lloyd et al., 2013)</td><td>One-step ahead Prediction</td><td>Multi-tier Applications Deployment Performance in Cloud</td><td>Multiple Linear Regression</td><td>Historical Data</td><td>Impact of Applications Deployment on Resource Usage and Performance</td></tr>
<tr><td>(Liang et al., 2011)</td><td>Long-term Prediction</td><td rowspan="2">CPU Load</td><td>Fourier Transform, Tendency-based Method to predict the variation</td><td>Seasonal Variation</td><td rowspan="2">Does Not Apply</td></tr>
<tr><td>(Hu et al., 2013)</td><td>Multi-step ahead Prediction</td><td>Support Vector Regression (SVR) and Smooth Kalman Filter</td><td>Non Adaptive to Unexpected Changes</td><td>Training Data (size of training data)</td></tr>
<tr><td>(Gambi et al., 2016)</td><td>Self-adaptive cloud controller</td><td>Cloud Controller</td><td>Kriging models</td><td>Self-adaptive</td><td></td><td></td><td>Minimization of SLA Violation</td><td></td></tr>
<tr><td>Our Approach</td><td>Online Multi-step ahead Prediction</td><td>Generic</td><td>Kriging Method and Time Series</td><td>Self-adaptive</td><td>Dynamic and Variable Sliding Window</td><td>Reactive Error Adjustment and Adaptive Padding</td><td>Reduction of under-provisioning</td><td>Reduction of over-provisioning</td></tr>
</table>

no assumptions or knowledge of the systems' internal functionalities being required. We also provide an adaptive solution capable of accommodating changes in the system behavior through real-time data analysis. Moreover, our solution provides multi-step ahead resource demand prediction by leveraging the Kriging machine learning method and time series, and proposing a dynamic sliding window technique. Further, this work presents a novel dynamic

adaptive padding and reactive error adjustment which can mitigate resource under-estimations and over-estimations to reduce SLA violations and reduce resource loss due to typical excessive allocation of resources.

2.4 Approach overview

In the present work, we propose a generic, dynamic, and self-adaptive prediction of the resource needs in virtualized systems. The proposition aims to minimize under-estimation, which can lead to possible SLA violations, and reduce over-estimation, which that causes a loss of resources, without any prior knowledge of the system or any assumption regarding its behavior or load profile. To that end, we propose a novel prediction algorithm that involves three main techniques. The first mechanism leverages the Kriging method for dynamic machine learning-based prediction. The second technique considers the input of the algorithm, namely, the resource utilization data collected from the system, as a time series with a variable sliding window. This technique benefits from Genetic Algorithm (GA) to dynamically provide the optimal size of the sliding window and the optimal number of predicted data, helping to minimize number prediction errors due to under- and over-estimation. This enables our algorithm to dynamically provide the prediction based on the resource utilization data collected in real time reflecting the current system state. The third and last technique adjusts the prediction based on the estimated probability of prediction errors and a variable padding. Figure 2.2 presents the main components of the proposed resource prediction algorithm.

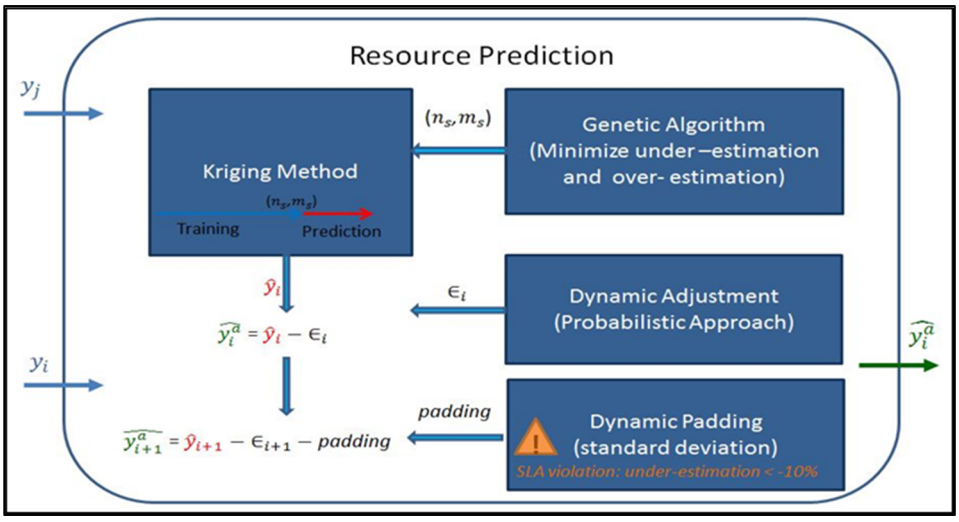

Figure 2.2 Resource prediction components

This section gives an overview of the proposed solution, while the details are left for the following sections. Table 2.2 describes the notations and the symbols used in the rest of the paper. Algorithm 2.1 presents the proposed approach for resource needs prediction. It starts by initializing the size of the sliding window and the number of predicted data (n_i, m_i) to their maximums (maximums of a given sets of values defined in configuration phase), while the error adjustment coefficient and the padding values are set to zero (Line 1 to Line 4). After resource utilization data is collected, an initialization phase (Line 5) is performed. It consists of consecutive training and prediction steps based on the Kriging method, in which we gather sufficient data (named historical data) to apply our techniques for adjustment and optimization. The prediction and its adjustment are applied based on the historical resource utilization data for each pair (n_i, m_i) of sets of all possible combinations of n_i, m_i values (Line 6 to Line 10). The results obtained, which form the adjusted predicted data and their corresponding combination of (n_i, m_i), are used by Genetic Algorithm (Algorithm 2.4) to determine the optimal sliding window and prediction sizes (n_s, m_s) that minimize under- estimation and over-estimation (Line 11). With the optimal pair (n_s, m_s) determined, the upcoming resource consumption prediction is performed based on the Kriging method (Line 12) and the adjustment algorithm (Line 13), using the two previous error adjustment values (Algorithm 2.2). Once the first resource utilization value is collected, it is compared to its corresponding

adjusted predicted data. If there is an under-estimation greater than 10%, the padding value is evaluated (Algorithm 2.3) and the prediction/adjustment processes are resumed, taking padding into account. We defined the threshold based on empirical studies. Otherwise, the resource utilization data are gathered for the next prediction step (Line 18). Our online prediction process continues to estimate the resource utilization repeatedly while the system is monitored and its relevant data are collected.

Algorithm 2.1 Resource Consumption Prediction (y_i, *n, m*)

Resource Consumption Prediction (y_i,*n*, *m*)
Input: (y_i,*n*, *m*)
Output: $\widehat{y_i^a}$
1: $(n_i, m_i) :=$ (max($\{n_i\}$), max ($\{m_i\}$))
2: Initialize error adjustment coefficient $\in_{i-1}$:=,$\in_{i-2}$:= 0
3: Initialize padding:=0
4: Collect observed data: y_i , $i \in [1..n_i]$
5: Initialization Phase (n_i, m_i)
6: ***for*** each collected data window
7: ***for*** *each* (n_i, m_i)
8: Predict historical data ($\hat{y}_i$, n_i, m_i) = Kriging (y_i, n_i, m_i)
9: Adjust historical data ($\widehat{y_i^a}$, n_i, m_i , $\in_{i-1}, \in_{i-2}$)= **Algorithm 2.2** ($\hat{y}_i, I_{i-2}$, I_{i-1}, $\in_{i-1}$, $\in_{i-2}$,)
10: ***end for***
11: Genetic Algorithm (n_s, m_s) = **Algorithm 2.4** ($\widehat{y_i^a}$, $\{(n_i, m_i)\}$)
12: **Predict next data (**$\hat{y}_i$, n_s, m_s**)= Kriging (**y_i, n_s, m_s**)**
13: ($\widehat{y_i^a}$, $n_s, m_s, \in_{i-1}$, $\in_{i-2}$) = **Algorithm 2.2** ($\hat{y}_i, I_{i-2}, I_{i-1}, \in_{i-1}, \in_{i-2}$)
14: Collect observed data: y_i
15: padding:= **Algorithm 2.3** ($\widehat{y_i^a}, y_i, n_s, m_s, threshold$)
16: return the adjusted prediction of resource utilization $\widehat{y_i^a}$
17: ***if*** (padding = =0)
18: Collect observed data: y_i , $i \in [i+1..i-1+m_s]$
19: ***else***
20: Go to step 7
21: ***end***
22: ***end for***

Table 2.2 Terms, definitions and symbols

Symbol /Acronym	**Definition**
y_i	Observed data
$\hat{y}_i$	Predicted data
$\widehat{y_i^a}$	Adjusted data
e_i	Error of the ith prediction: $e_i = \hat{y}_i - y_i$
X	Continuous random variable that represents the observed error e_i
I	Interval of errors $I = [e_{min}, e_{max}]$ for each prediction step
PDF	Probability Density Function (e.g., Normal, non-parametric)
$Pr(x \in I)$	Probability that X is in the interval I.
I_{proba}	Interval of probability (e.g., $I_{Proba} = [0,0.1[$)
$\in_i$	Error adjustment coefficient (e.g., min or max of errors) in the interval I_i
l	Number of sliding windows
n_{oe}	Number of over-estimations
n_{ue}	Number of under-estimations
α_i	Indicates whether or not a padding is added
β_i	Indicates whether or not there is an over-estimation
γ_i	Indicates whether or not the size of the sliding window is applicable
m_i	Number of predicted data in the interval I_i
n_i	Number of observed data within a sliding window used for training data in the interval I_i
n_s	Optimal number of training data in the interval I_i
m_s	Optimal number of predicted data in the interval I_i
r	rounded ratio between the observed and the adjusted data: $r = \lceil y_i / \widehat{y_i^a} \rceil$
σ_j	Standard deviation: $\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2}$ of the jthunder-estimation if it is less than -10%
$\bar{y}$	$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$
$Pr_{UnderEstim}^{PredictData}$	Probability of under-estimation for predicted data: $\frac{number\ of\ underestimations\ in\ predicted\ data}{number\ of\ predicted\ data}$
$Pr_{UnderEstim}^{AdjustData}$	Probability of under-estimation for adjusted data: $\frac{number\ of\ underestimations\ in\ adjusted\ data}{number\ of\ adjusted\ data}$
$E_{OverEstim}^{PredictData}$	Mean of over-estimation for predicted data: $\frac{1}{n_{oe}}\sum_{i=1}^{n}(\hat{y}_i - y_i)$
$E_{OverEstim}^{AdjustData}$	Mean of over-estimation for adjusted data: $\frac{1}{n_{oe}}\sum_{i=1}^{n}(\widehat{y_i^a} - y_i)$
$E_{OverEstim}^{Thres}$	Mean of over-estimation for threshold-based provisioning $E_{OverEstim}^{Thres} = \frac{1}{n_{oe}}\sum_{i=1}^{n}(Thres - y_i)$
$Thres$	Over-provisioning of resources in legacy networks (maximum allocated resources).
$uptime_i$	The ith uptime moment where there is no under-estimation.
$downtime_i$	The ith downtime moment where there is under-estimation after the ith uptime moment.
$MTBUE$	Mean Time Between Under-Estimations: $\frac{\sum_{i=1}^{n}(uptime_i - downtime_i)}{number\ of\ under-estimations}$
CPS	Call Per Second
MSE	Mean Squared Error = $\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$

Algorithm 2.1 Complexity: The time complexity of the proposed resource consumption prediction algorithm depends essentially on three parts: time taken by the Kriging method to train and predict the next resource demand, the time complexity of adjustment and padding, and the time complexity of GA to provide the optimal size of the sliding window and the number of predicted data (n_s, m_s). The time complexities of each part of our algorithm, namely, the Kriging method, adjustment, padding and GA, are evaluated in Sections 4, 5, 6,and 7, respectively. In Algorithm 2.1, the initialization of parameters (the sliding window size, the number of predicted data, the error adjustment value, the padding value), as well as the resource utilization data collection (Line 1-Line 4) have a time complexity $O(1)$. During the initialization phase (Line 5), several training and prediction steps using the Kriging method are performed with a time complexity of $O(k\ m_i\ n_i^3)$, where k is the number of repetitions used to collect sufficient data for adjustment and optimization. Then, the assessment of the optimal (n_s, m_s) is performed using GA (Line 11), based on the results of the prediction and adjustment of the historical data (Line 7-10). These two steps have time complexities of $O(IPL)$ and $O(P\ m_i\ n_i^3) + O(PNI)$, respectively, where P is the size of the population of (n_i, m_i). The prediction (Line 12), adjustment of predicted resource demands (Line 13), and padding (Line 15) have time complexities of $O(m_s n_s^3)$, $O(NI)$, and $O(n_s)$, respectively. Finally, data collection, the evaluation of padding values and provision of the estimation of resource demands have a time complexity of $O(1)$. Consequently, the time complexity of Algorithm 2.1 is $O(1)+ O(k\ m_i\ n_i^3) + O(P\ m_i\ n_i^3) + O(PNI) + O(IPL) + O(m_s n_s^3) + O(NI)+O(1)+ O(n_s) + O(1)$, which is equivalent to $O(P\ m_i\ n_i^3)$ due to the highest order of n_i^3.

2.5 Prediction

Kriging (Krige, 1951) (Matheron, 1963) is a spatial interpolation procedure that uses statistical methods for prediction. It assumes a spatial correlation between observed data. In other words, observed data close to each other in the input space are assumed to have similar output values (Gambi et al., 2016). Kriging is able to model a system based on its external behavior (black-box model) and generic data. It also provides adaptability to linear, non-linear and multi-modal behaviors of the system (i.e., runtime training), with a complexity that varies with the number

of samples used for the model fitting. These characteristics are exactly what make the Kriging method suitable for online adaptive and dynamic prediction, specifically in the cloud context (workload variability), which has also been proved in the literature (Gambi et al., 2016). In this work, we adapt Kriging in order to provide dynamic and adaptive resource consumption prediction. Next, we explain how the proposed method works.

Through interpolation, the method predicts the unknown value $\hat{y}_p$ by computing weighted linear combinations of the available samples of observed data y_i in the neighborhood given in Equation 2.1 (Van Beers & Kleijnen, 2004)

$$\widehat{y_p} = \sum_{i=1}^{n} \omega_i \, y_i \tag{2.1}$$

where ω_i is the weight associated with the estimation, and $\sum_{i=1}^{n} \omega_i = 1$.

To quantify the weight ω_i for each observed datum (Equation 2.1), the method determines the degree of similarity between the observed data, from the covariance value, according to the distance between the sample points (y_i , y_j), using the semivariogram $\gamma(h)$ given by (Gratton, 2002) (G. Liu et al., 2014) :

$$\gamma(h) = \frac{1}{2N(h)} \sum_{i=1}^{N(h)} (y_i - y_j)^2 \tag{2.2}$$

where *N(h)* is the number of all pairs of sample points (y_i , y_j) (i.e., observed data) separated by the distance *h*.

The empirical semivariogram allows deriving a semivariogram model (e.g., Spherical, Gaussian, Exponential) to represent the semivariance as a function of separation distance. The semivariogram model is used to define the weights ω_i and to evaluate the interpolated points (i.e., predicted data). Hence, the weights ω_i are obtained by resolving the following linear equation system (Gratton, 2002) (G. Liu et al., 2014):

$$\begin{bmatrix} A & 1 \\ 1^T & 0 \end{bmatrix} \begin{bmatrix} \vec{\omega} \\ \mu \end{bmatrix} = \begin{bmatrix} B \\ 1 \end{bmatrix} \quad (2.3)$$

where $A_{ij} = \gamma(h_{ij})$ is a value of semivariogram corresponding to distance h_{ij} between y_i and y_j, $B_{ip} = \gamma(h_{ip})$ is the value of semivariogram to be calculated according to the distance h_{ip} between y_i and y_p (point to estimate), $\vec{\omega} = [\omega_1, \ldots, \omega_n]^T$ is the weight vector, and μ is the Lagrange multiplier.

Finally, the calculated weights are used in Equation 2.1 to estimate y_p.

In our proposition, we use the Kriging method to predict the next m_i values of CPU consumption ($\widehat{y_p}$ in Equation (2.1)) using n_i observed values of CPU consumption (y_i in Equation (2.1)) as training data. To estimate resource needs for multi-step ahead prediction, the method determines the weights of observed CPU consumption data (training phase) by solving the linear system in Equation (2.3), which has a time complexity of $O(n_s^3)$, with n_s being the number of training data (Srinivasan, Duraiswami, & Murtugudde, 2008). Hence, at each prediction phase, the Kriging method has a time complexity of $O(m_s\, n_s^3)$, with m_s being the number of predicted values. A complexity of $O(m_s\, n_s^3)$ is acceptable as the size of the training data (sliding window) and the numbers of the predicted data are variable and relatively small values for use in closely tracking the system behavior. Indeed, we conducted divers experiments on the impact of training/prediction window size variation on prediction performance. The results show an increase in prediction errors when the training data size increases specifically for fluctuating workloads. This is due to real-time data-driven algorithm process and its need for continuous adaptation to changes in system resource demands. Therefore, the sizes of training and predicted data are relatively small even with GA optimization (n_i , $m_i \in [3,20]$ in our experiments) and the time complexity of the prediction decrease consequently.

2.6 Adjustment

To improve the efficiency of the prediction method and reduce the under-estimation caused by significant changes in resource demand, we propose a dynamic prediction adjustment strategy based on the estimated probability of prediction errors (Algorithm 2.2) and a variable padding technique. Algorithm 2.2 describes the proposed strategy. We determine the error adjustment coefficient $\in_i$ that reflects the current tendency for under/over-estimation and we add it to the predicted data. In the event of a significant under-estimation, particularly one over the given tolerance threshold (Algorithm 2.3), padding is added to the adjusted predicted data in order to prevent critical under-estimation and SLA violations (Algorithm 2.2, line 14). We set tolerance threshold to 10% because further experiments showed that thresholds below 10% induced excessive padding and consequently an over-estimation of resource needs. Otherwise, the padding value is null.

In our probabilistic approach, we consider the prediction error e_i as a continuous random variable, denoted by *X*. Its probability density function (PDF), $\rho(x)$, defines a probability distribution for *X* (Dunn, 2014). The probability that *X* will be in interval I, with $I = [x_1\ ,\ x_2]$, is given by:

$$\Pr(x\ \in I) = \int_I \rho(x)dx = \int_{x_1}^{x_2} \rho(x)dx \tag{2.4}$$

with $\rho(x) \geq 0$ for all *x* and $\int \rho(x)\ dx = 1$

Based on historical data, we set the *I* as an interval of values between the minimum and the maximum of previously observed errors; $I = [e_{min}\ ,\ e_{max}]$. Additionally, we define two probability intervals I_{Proba} and $\acute{I}_{Proba}$ ($I_{Proba} = [0{,}0.1[$ and $\acute{I}_{Proba} = [0.1{,}1]$) and we assume that: the PDF is Gaussian (most common PDF for continuous process); and an error e_i is more likely if its probability $\Pr(x\ \in I)$ belongs to $[0.1{,}1]$ and is less likely if its probability $\Pr(x\ \in I)$ belongs to $[0{,}0.1[$.

Each time a prediction is made, the probabilities of two previous error intervals $Pr_{i-2}(x \in I_{i-2})$ and $Pr_{i-1}(x \in I_{i-1})$, are compared, along with the error indicators (i.e., under-estimation if $e_i < 0$; over-estimation otherwise) (Line 2). If the two probabilities belong to the same probability interval (I_{Proba}) and they have the same indicators, we assume that the system may have a stable tendency and the current prediction is adjusted by the maximum of the previous errors (Line 3). Otherwise, we assume that there is a change in the workload and/or in the system behavior, and hence the current prediction is adjusted either by the minimum of the two previous error adjustment coefficients (if they are positives, which denote two consecutive over-estimations, in order to minimize the over-estimation (Lines 5-6), or by the most recent error adjustment coefficient in order to track the change (Lines 7-8). It is relevant to mention that the number of error intervals considered in prediction adjustment was dictated by several executions of Algorithm 2.2 to find the best adjustment.

Algorithm 2.2 Adjustment of the Prediction ($\hat{y}_i, I_{i-2}, I_{i-1}, \in_{i-1}, \in_{i-2}$,)

Adjustment of the Prediction ($\hat{y}_i, I_{i-2}, I_{i-1}, \in_{i-1}, \in_{i-2}$,)
Input: ($\hat{y}_i, I_{i-2}, I_{i-1}, \in_{i-1}, \in_{i-2}$,)
Output: ($\widehat{y_i^a}$, $n_s, m_s, \in_{i-1}, \in_{i-2}$)

1: Compute $Pr_{i-2}(x \in I_{i-2})$ and $Pr_{i-1}(x \in I_{i-1})$
2: ***if*** $\{Pr_{i-1}(x \in I_{i-1})$, $Pr_{i-2}(x \in I_{i-2})\} \in I_{Proba}$ and sign($\in_{i-1}$)= sign($\in_{i-2}$)
3: $\in_i$:= maximum (| $\in_{i-1}$ |, | $\in_{i-2}$ |) × sign ($\in_{i-1}$)
4: ***else***
5: ***if (***$\in_{i-1} > 0$ and $\in_{i-2} > 0$)
6: $\in_i$:= minimum ($\in_{i-1}$, $\in_{i-2}$)
7: ***else***
8: $\in_i$:=$\in_{i-1}$
9: ***end***
10: ***end***
11: ***if*** ($\in_i > 0$)
12: $\widehat{y_i^a} = \hat{y}_i + \in_i$
13: ***else***
14: $\widehat{y_i^a} = \hat{y}_i - \in_i$- padding
15: ***end***
16: ***return*** ($\widehat{y_i^a}$, $n_s, m_s, \in_{i-1}, \in_{i-2}$)

The time complexity of the prediction adjustment (Algorithm 2.2) is influenced by the evaluation of the probabilities of the two previous error intervals (Line 1) using a numerical integration of the probability density function (Eq. (2.4)). The integral is computed via Simpson's rule (Abramowitz & Stegun, 1972) (MathWorks, 2017b) (MathWorks, 2017a) in the interval $[e_{min}, e_{max}]$ with N_I equally spaced points, which is performed with a time complexity of $O(N_I)$. The calculation of the error adjustment coefficient $\in_i$ (Line 2-10) has a time complexity of $O(1)$. Also, the calculation of adjusted data (Lines 11-15) and its return (Line 16) both have a time complexity of $O(1)$. Hence, the time complexity of the prediction adjustment algorithm is $O(N_I).+O(N_I)+O(1)+O(1)+O(1)$, which is equivalent to $O(N_I)$.

2.7 Padding strategies

If the under-estimation exceeds a tolerance threshold (e.g., 10%), an additional adjustment, called padding, is computed (Algorithm 2.3) and added to the adjusted predicted data in the next prediction step in order to quickly address the gap between the observed data and the predicted data. This ultimately prevents a long-lasting under-estimation and SLA violations. In this context, we tested two padding strategies, namely, ratio-based and standard deviation-based strategies in order to measure the amplitude of variation in resource consumption and adjust the next prediction results accordingly. The first strategy is based on a ratio r between the observed data and the adjusted data, and the error between them (Equation 2.5):

$$padding = r\,(\widehat{y_i^a} - y_i) \tag{2.5}$$

where $r = \lceil y_i \,/\, \widehat{y_i^a} \rceil$

This ratio-based padding shows large over-estimations when significant fluctuations, such as a sharp increase followed by a sharp decrease, occur in the workload. Therefore, we propose another padding strategy that considers workload variability. It is based on the standard deviation of the previous observed data. The mean of previous standard deviations of observed data is considered as a value of the padding (Equation 2.6):

$$padding = (mean\ (\sigma_j\ (y_{i-n_s}, y_i)) \tag{2.6}$$

where $j \in \{1,...,l\}$, l is the number of under-estimations greater than 10% and n_s is the optimal number of training data in the interval I_i

The time complexity of the padding in Algorithm 2.3 depends on the computation of standard deviation of the previous observed data (Line 2), which is $O(n_s)$, and the mean of previous standard deviations of observed data (Line 4 or Line 6), corresponding to $O(l)$. The rest of the statements in this algorithm each have a time complexity of $O(1)$. Hence, the time complexity of the padding algorithm is $O(1)+O(n_s)+ O(1)+O(l)+O(1)+O(1)$, which is equivalent to $O(n_s)$ having $n_s > l$.

Algorithm 2.3 Padding $(\widehat{y_i^a}, y_i, n_s, m_s, threshold)$

Padding $(\widehat{y_i^a}, y_i, n_s, m_s, threshold)$
Input: $(\widehat{y_i^a}, y_i, n_s, m_s, threshold)$
Output: *padding*
1: ***if*** $(\widehat{y_i^a} < y_i)$ ***and*** $(\frac{\widehat{y_i^a} - y_i}{y_i} < threshold)$
2: $\sigma_{current} = \sigma_j\ (y_{i-n_s}, y_i)$
3: ***if*** $\sigma_{current} > 2\sigma_{previous}$
4: *padding = mean* $(\sigma_j, j \in \{1,...,l\text{-}1\})$
5: ***else***
6: *padding = mean* $(\sigma_j, j \in \{1,...,l\})$
7: ***end***
8: ***else***
9: *padding = 0*
10: ***end***
11: ***return*** *padding*

2.8 Optimization

A time series is an ordered collection of values obtained through repeated measurements, typically over equally-spaced time intervals (W. S. Wei, 1990), and its analysis allows the extraction of a model that describes particular patterns in the collected data. Therefore, for dynamic resource consumption prediction, we consider real-time data collected from the system as time series, where at each sliding window, the prediction of the next resource demand is performed. Each set of *i* observed data is used in the training phase of the prediction model in order to predict the next *j* values. Afterwards, the sliding window is slid forward by *j* values at each prediction step in order to continuously keep track of, and predict, the system resource usage.

With the efficiency of the prediction model being mainly affected by the size of the sliding window, as a first step, we studied the performance of the typical fixed sliding window strategy. We tested the prediction model with respect to the mean absolute percentage error (MAPE) metric. Next, experiments were carried out with different sliding windows, by varying both training data and predicted data numbers (e.g., (13-20), (10-15), (7-10), respectively), and the ratio between them. Although the fixed sliding window strategy was able to provide efficient prediction results with MAPE < 10%, a prior testing and evaluation phase was required to determine the best pair (n_s, m_s). Furthermore, the results showed critical performance degradation when abnormal system behaviors were observed. Abnormal behavior includes, for instance, a sharp workload increase or decrease. Our observations showed that an efficient and adaptive prediction, which is able to deal with typical and unpredictable system behaviors, is a function of both the size of the sliding window and the number of predicted samples. Both parameters have direct impacts on the accuracy of the prediction, particularly when there are significant fluctuations in the workload. Therefore, the main objective in this section is to find the best sliding window size (n_s) and the best number of predicted data (m_s) that minimize resource demand over-estimation and under-estimation before each prediction process.

2.8.1 Problem formulation

We define these goals in a multi-objective optimization problem which we formulate as follows:

- Minimize over-estimation: The mean of over-estimations is equal to the total of over-estimations divided by the number of over-estimation occurrences in historical data. An adjusted predicted datum is considered as an over-estimation if it is greater than its corresponding observed data.

$$F1=min\ (\frac{1}{n_{oe}}\ \sum_{k=1}^{l}\sum_{j=1}^{n}\gamma_i(\sum_{i=1}^{m}[(\hat{y}_{ijk}+\varepsilon_{ijk})-y_{ijk}]\beta_i)) \quad (2.7)$$

- Minimize under-estimation: The mean of under-estimations is equal to the total of under-estimations divided by the number of under-estimation occurrences in historical data. An adjusted predicted datum is considered as an under-estimation if it is less than its corresponding observed data.

$$F2=min\ (\frac{1}{n_{ue}}\ \sum_{k=1}^{l}\sum_{j=1}^{n}\gamma_i(\sum_{i=1}^{m}[(\hat{y}_{ijk}-\varepsilon_{ijk}-\alpha_i padding)-y_{ijk}](1-\beta_i))) \quad (2.8)$$

Thus, our multi-objective optimization problem is:

$$F = min\ \{F_1, F_2\} \quad (2.9)$$

subject to

$$n \in \mathbb{N} \quad \text{(c1)}$$
$$m \in \mathbb{N} \quad \text{(c2)}$$
$$m \leq n \quad \text{(c3)}$$
$$\alpha_i = \{0,1\} \quad \text{(c4)}$$
$$\beta_i = \{0,1\} \quad \text{(c5)}$$
$$\gamma_i = \{0,1\} \quad \text{(c6)}$$

These objective functions aim to minimize resource wastage and SLA violations, respectively. The constraints c1, c2 ensure that the sizes of data and the sliding window belong to the natural

numbers, while c3 confirms that the number of predicted data is less than or equal to the sliding window size. Finally, constraints c4, c5 and c6 define the decision variables for padding, over-estimation and the sliding window size, respectively, as binary variables. The solution of this problem is the best combination of sliding window size (n_s) and the predicted data number (m_s) which minimizes over-estimation and under-estimation of the resource requirements, allows dynamic prediction, and improves the performance of the latter.

2.8.2 Heuristic algorithm-Genetic Algorithm

Genetic Algorithms (GAs) (Tout et al., 2019) (Mitsuo & Runwei, 2000) (X.-S. Yang, 2010) form an abstraction of biological evolution, which imitate the natural evolution process to solve an optimization problem. GAs are heuristic methods that aim to determine the best solution by simulating the propagation of the fittest individuals over consecutive generations. The fitness of a solution is based on a function that aims to minimize or maximize (a) particular objective(s). Through crossover, mutation and selection operators, Genetic Algorithm is able to generate diversified individuals and find the best among them. This approach might be a good alternative to an exhaustive search, as shown in Section 2.8.

In this paper, we use GA to determine the optimal size of the sliding window and the optimal number of predicted data *(n_s, m_s)*. Hereafter, we explain how we adapted GA to solve our optimization problem defined in the previous section.

2.8.2.1 Individuals of populations

Each individual in the populations is represented by two elements. The first element is a size of the sliding window n_i, and the second one is a size of the predicted data m_i. The set of (n_i, m_i) combinations constitutes a population. For instance, the set {(5, 3), (10, 3), (10, 7), (15, 6), (15, 10), (20, 4)} represents a population of six different individuals.

2.8.2.2 Fitness function and selection

The fitness function in GA aims to assign a score to each individual according to its efficiency in resolving the problem. In this work, the score of a solution is computed by evaluating the objective functions F1 and F2. Next, the selection of individuals is carried out by evaluating their fitness. Individuals with a higher fitness are more likely to reproduce in the next generations.

2.8.2.3 Crossover and Mutation

Next, crossover and mutation operators are applied with rc. and rm. rates, respectively. Typically, the crossover of two individuals can be conducted by swapping elements from both individuals, resulting in two offspring. For instance, individuals (10, 3) and (20, 7) can be crossed over each other's first element to generate the two offspring (20, 3) and (10, 7). Alternatively, mutation can be realized by randomly flipping an element in different individuals, selected based on rm. For example, an individual (10, 7) could be mutated in its second element to yield (10, 2).

2.8.2.4 Algorithm 2.4 - Genetic Algorithm

All the steps of the adapted GA are described in Algorithm 2.4. It starts by generating the initial population. The fitness of each individual is computed by evaluating both predefined objective functions that aim to minimize under- and over-estimations. The fittest individuals in the population are selected and inserted into the next population. Crossover and mutation operators are applied to produce a new generation of solutions (n_s, m_s), and the fitness of these individuals is calculated. This process is repeated until the stopping condition is met. The latter can be defined as the time constraint, number of generations, or any other adequate criterion. Finally, the fittest pairs (n_s, m_s) from the last generation are reported. If several solutions are provided by GA, the solution that most minimizes the under-estimation is selected for the next prediction because we consider that the under-estimation is more critical than the over-estimation in terms of cost of SLA violations.

Algorithm 2.4 Genetic Algorithm($\widehat{y_i^a}$, $\{(n_i, m_i)\}$)
Adapted from (Tout et al., 2019)

Genetic Algorithm($\widehat{y_i^a}$, $\{(n_i, m_i)\}$) – adapted from (Tout et al., 2016)
Input: ($\widehat{y_i^a}$,, $\{(n_i, m_i)\}$)
Output: (n_s, m_s)

1: Initialize N = population size; r_c = crossover rate; r_m = mutation rate
2: Initialize population index $j = 0$
3: Generate the initial population P_j
4: ***for*** each individual (n_i, m_i) in P_j
5: Evaluate objective functions $F_1(\widehat{y_i^a}, (n_i, m_i))$, $F_2(\widehat{y_i^a}, (n_i, m_i))$
6: ***end for***
7: ***do***
8: Select x best (n_i, m_i) and insert them into P_{j+1}
9: Crossover $r_c \times n$ individuals to produce new offspring (n_i, m_i) and insert them into P_{j+1}
10: Mutate $r_m \times n$ individuals to produce new offspring (n_i, m_i) and insert them into P_{j+1}
11: ***for*** each individual (n_i, m_i) in P_{j+1}
12: Evaluate objective functions $F_1(\widehat{y_i^a}, (n_i, m_i))$, $F_2(\widehat{y_i^a}, (n_i, m_i))$
13: ***end for***
14: j= j+1
15: ***while*** stopping criteria is not met
16: ***return*** the fittest (n_s, m_s) from P_j

2.9 Experimental evaluation

We evaluated the cost of the resource demand prediction in terms of SLA violation and resource wastage by computing the probability of under-estimations (e.g., $Pr_{UnderEstim}^{PredictData}$), the mean of over-estimations (e.g., $E_{OverEstim}^{PredictData}$) and the mean time between under-estimations (*MTBUE*) for both predicted and adjusted data. Also, we considered the mean of over-estimations in the case of static provisioning of resources (threshold-based provisioning), that is, an over-provisioning of resources applied in legacy systems. It represents the maximum of allocated resources for a specific system and load profile. We use this metric to compare the

resource gains between our approach (prediction and adjustment) and the static provisioning approach. Details of the calculation of suggested metrics are presented in Table 2.2.

To evaluate our algorithm with different types of virtualized systems/applications, we used data from the OpenIMS telecommunication service platform (Configurations 1 and 2 presented in Table 2.4), data furnished by an IMS service provider (Datasets 1 and 2), and data extracted from Google cluster data available online (Reiss, Wilkes, & Hellerstein, 2014) (Datasets 3 and 4). By processing data from various systems and load profiles, we aim to evaluate the ability of our algorithm to accurately predict the resource utilization within various types of systems/applications, workloads and situations (predictable vs. unpredictable workloads).

2.9.1 Experimental Setting

2.9.1.1 Testbed setup

The testbed was built with 4 servers, all connected to the same local area network. For IMS Core components, we used OpenIMS Core (Fokus, 2014), which is an open source implementation of IMS Core developed by Fraunhofer FOKUS IMS Bench. The virtualization of IMS Core (CSCFs and HSS) is based on Linux container technology (Containers, 2017). Further, a SIP client was created using the instantiation of SIPp version 591 (October 2010), which is a traffic generator (Gayraud & Jacques, 2014). The Virtualized P-CSCF and HSS were hosted on Server 1, whereas, the Virtualized S-CSCF and I-CSCF were hosted on Server 2. The high-level monitoring of virtualized CSCF entities and the management of resources was done by the Host Monitoring Agent (HMA) and Resource Management Agent (RMA), respectively. HMA and RMA were deployed on both Server 1 and Server 2. Server 3 hosted SIPp and the Session Monitoring Engine (SME). DNS (Domain Name System) was hosted on Server 4. Each physical machine had an Intel Core i7 3.6 GHz CPU, 24.576 GB of RAM, and a Linux Ubuntu operating system version 14.04. Figure 2.3 presents the testbed.

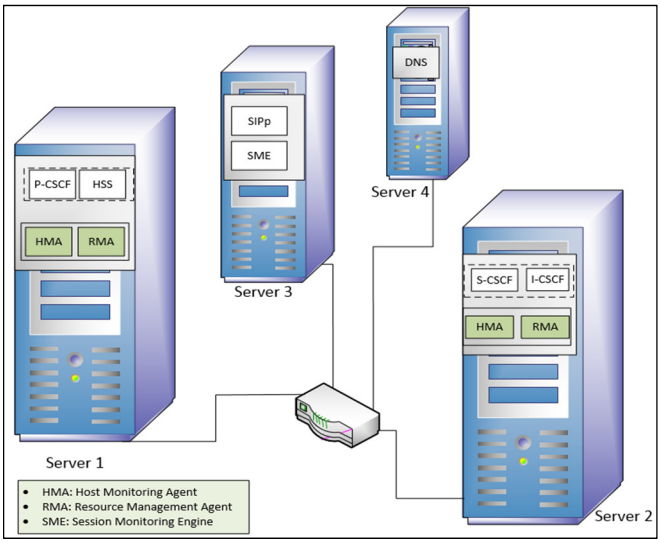

Figure 2.3 Testbed

For the parameters of the Kriging method, we essentially set its type to universal Kriging, and the variogram model to the spherical one, with a range of 0.2 and a sill of 1. These parameters were set through tests trying to find the configuration minimizing prediction errors.

Regarding the GA configuration, we used the values presented in Table 2.3. The population was initialized as {(10,3), (10,5), (10,7), (20,5),(20,10), (20,14)}, the selection was based on the tournament selection function (Mitsuo & Runwei, 2000), the single-point crossover was used, while a random modification of n_i or m_i values was performed for the mutation. The crossover and the mutation probabilities were set to 0.8 and 0.2, respectively. Finally, the stopping criterion was set to the number of iterations. We set the value of this criterion based on multiple executions of this algorithm while trying to find the optimal pair (n_s, m_s).

2.9.1.2 Data /Load profile

We focused our tests on the CPU consumption of the virtualized S-CSCF node because the CPU load has a significant effect on the performance (Liang et al., 2011), and a performance analysis of OpenIMS showed that S-CSCF is a bottleneck in IMS (Mkwawa & Kouvatsos, 2008). Then, we performed several tests on the OpenIMS platform using different load profiles, representing for instance, a sharp decrease (Configuration1 (Conf1)), or a sharp decrease

Table 2.3 Parameters of Genetic algorithm

Parameter	**Value**
Population size	6 pairs (n_i, m_i)
Initial Population	{(10,3), (10,5), (10,7), (20,5),(20,10), (20,14)}
Selection	Tournament selection function
Crossover probability	0.8
Mutation probability	0.2
Crossover operator	Single point
Mutation operator	Random modification of n_i *or* m_i values
Number of generations	201
Stopping criteria	Number of generations

followed by a sharp increase (Configuration2 (Conf2)) of the workload. The data from the OpenIMS platform was collected every 5 seconds.

The data from the IMS service provider represents CPU consumption collected from multiple machines every 5 minutes for 16 hours. In the present article, we present two examples of datasets (different numbers and amplitudes of workload fluctuations), namely, Dataset1 (Dset1) and Dataset2 (Dset2).

The Google cluster data trace (clusterdata-2011-2) represents 29 days' worth of cell information from May 2011, on a cluster of about 12,500 machines (packed into racks, and connected by a high-bandwidth cluster network). A cell is a set of machines, typically all in a single cluster, that share a common cluster-management system which allocates jobs to the machines. A job is comprised of one or more tasks (accompanied by a set of resource requirements) (Reiss et al., 2014). Because of the large size of the Google cluster data, we propose to extract the CPU consumption of tasks from multiple data files for a given machine and a given job. For instance, we present Dataset3 (Dset3) and Dataset4 (Dset4) that denote the CPU consumption of tasks identified as 85 and 42, respectively, collected every 5 minutes.

The descriptions of the load profiles, Configuration1 and Configuration2, are presented in Table 2.4.

Table 2.4 Description of load profiles (OpenIMS platform)

Load Profile	Description
Configuration1 (Conf1)	Start at 150 CPS, increment: 50 CPS/10 sec until 400 CPS, 400 CPS constant during 100 sec, 600 CPS constant for 300 sec, 200 CPS decrement: 50 CPS/50 sec until 50 CPS
Configuration2 (Conf2)	Start at 150 CPS, increment: 50 CPS/10 sec until 400 CPS, 400 CPS constant during 10 sec, 600 CPS constant for 300 sec, 5 CPS during 60 sec, 400 CPS constant for 300 sec

2.9.1.3 Assumptions

1. The monitoring tools are sufficiently accurate.
2. The monitoring and collection of data from the system are continuous and in real time.
3. The Linux container-based virtualization and the CPU core isolation technique guarantee that each service/application cannot access others' hardware and software resources. They also ensure that collected data reflect the effective service/application consumption.

2.9.2 Results and analysis

In this section, we present the evaluation of the ability of our algorithm to accurately estimate resource consumption, as well as the efficiency of the proposed techniques. First, we used two machine learning methods, namely, Kriging and SVR, to predict CPU consumption, and we compared their performance. Next, we evaluated the proposed prediction with adjustment and padding by processing data from various systems and assessed the cost in terms of SLA violation and resource wastage.

2.9.2.1 Prediction using Kriging vs. SVR methods

Beside the Kriging method, we also tested support vector regression (SVR) to predict resource consumption. We then compared their performance in terms of prediction errors and execution time. The Kriging is a spatial interpolation-based method, while SVR is a kernel-based learning method. SVR identifies the optimal hyperplane that maximizes the margin between the vectors (support vectors) of the considered classes. This optimal hyperplane is defined as a linear

decision function (finds optimal parameters w and b) (Cortes & Vapnik, 1995) (Smola & Schölkopf, 2004):

$$f(x) = wK(x) + b \tag{2.10}$$

where

x is input data, w is the weight vector, and b is the bias parameter

$K(x)$ is a kernel function (e.g., linear, radial basis function (RBF))

When the kernel function is set to *Radial Basis Function* (RBF), two parameters must be considered, namely, *C* and *gamma*. The parameter *C*, common to all SVR kernels, trades off misclassification of training samples against simplicity of the decision surface. The parameter *gamma* defines how much influence a single training sample has (developers Scikit-learn, 2013).

In our experiments, we used the CPU consumption data from OpenIMS platform tests (using different load profiles), and data from an IMS service provider and Google cluster data trace, which present several types of variations/fluctuations. Then, we performed prediction (without adjustment) with a fixed-size sliding window and a fixed number of predicted data: $(n_i, m_i) = (10,5)$ in the case of Kriging and $(n_i, m_i) = (10,10)$ in the case of SVR. The SVR kernel function was set to RBF since it is more appropriate for nonlinear datasets, while *C* and *gamma* values were set to 0.1. These parameters were set following extensive testing performed to find the configuration minimizing the prediction error.

In the first part of the experiments, we used the Kriging and SVR methods to predict CPU consumption for several configurations and datasets, and then compared their performance in terms of MSE (Mean Squared Error) and execution time. In the second part of the experiments, we tested the SVR method by varying the training and prediction data size and using Configuration 1 data to evaluate their impact on prediction performance. Figure 2.4 presents the results for each dataset, while Table 2.5 illustrates the MSE and the execution time of the considered datasets.

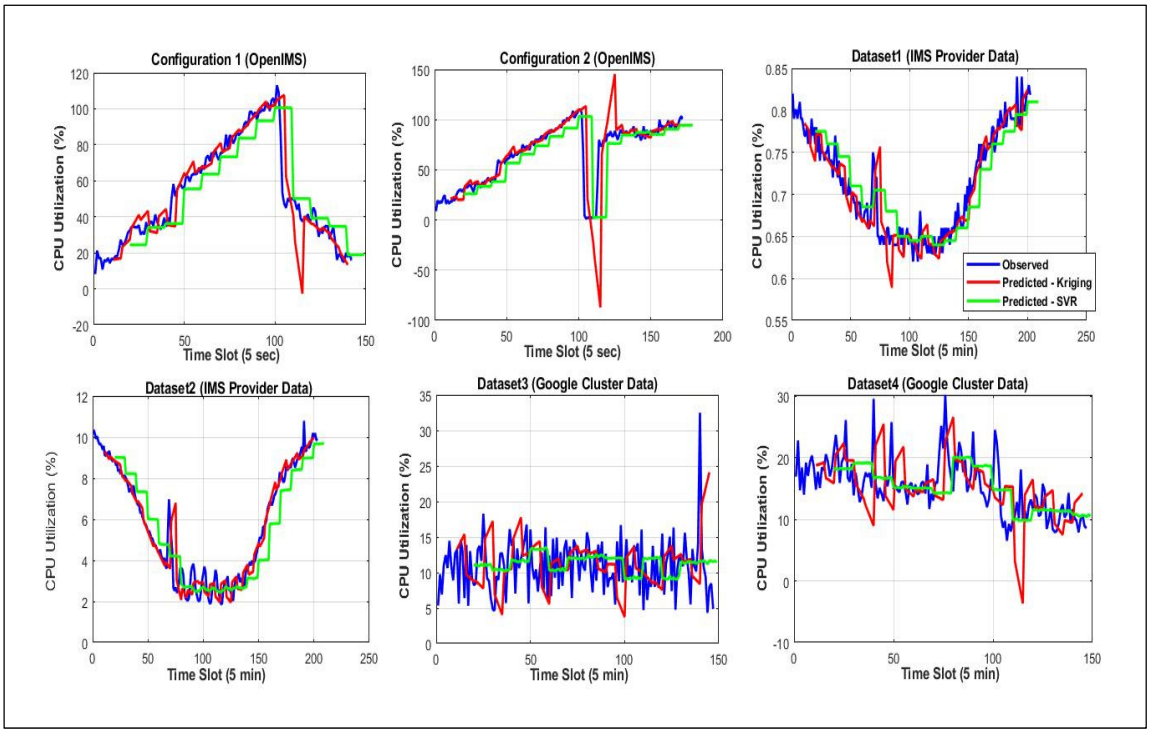

Figure 2.4 Prediction of CPU consumption for OpenIMS, IMS and Google cluster data: Kriging vs. SVR

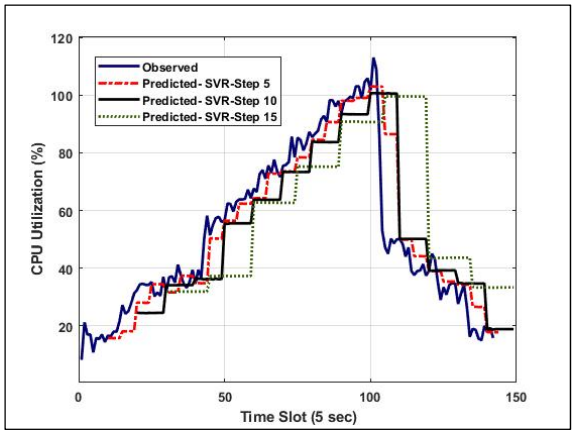

Figure 2.5 SVR prediction of CPU consumption in case of Configuration 1: variation of predicted data size

The results indicate that the Kriging method outperforms the SVR method in terms of prediction error (MSE), except for Dataset3 and Dataset4 (data from Google cluster). Indeed, Figure. 2.4 shows that the predicted CPU consumption using the Kriging method fits the observed workload better than the SVR method, with the exception of some areas where there are sharp CPU usage increases/decreases. For instance, the CPU usage prediction error was 645.51 in the case of Kriging, while it was 819.04 in the case of SVR for Configuration2 (see Table 2.5).

Table 2.5 MSE and execution time of Kriging vs. SVR prediction

Configuration/ Dataset	**Kriging method**		**SVR method**	
	MSE[a]	**Execution time (sec)**	**MSE**	**Execution time (sec)**
Configuration1	**123.40**	0.53	253.33	0.004
Configuration2	**645.51**	0.47	819.04	0.005
Dataset1	**5.78 10^{-4}**	0.55	7.7 10^{-4}	0.004
Dataset2	**0.58**	0.57	1.01	0.004
Dataset3	30.15	0.43	**15.95**	0.003
Dataset4	30.96	0.44	**15.72**	0.003

a: MSE: Mean Squared Error

Table 2.6 MSE and execution time of SVR prediction
-variation of training and predicted data size

Training/prediction data size	**SVR Method**	
	MSE	**Execution time (sec)**
3	82.80	0.01
5	144.54	0.007
10	253.33	0.004
12	493.53	0.003
15	664.68	0.002

On the other hand, we notice that the Kriging prediction error is greater than the SVR prediction error in the presence of sharper variations in short bursts, as observed in Dataset3 and Dataset4. The results also show that the SVR method is faster than the Kriging method (see Table 2.5, execution time). Furthermore, we observe (see Figure 2.5) that the prediction

error (MSE) increases when the training/prediction data size increases in the case of the SVR method (see Table 2.6). The latter is then sensitive to training/prediction data size, which is a major drawback, especially in the context of the dynamic sliding window.

To summarize, these experiments show that the Kriging method is able to accurately predict CPU consumption, but at higher cost in terms of execution time compared to SVR. Furthermore, the latter loses efficiency proportionally to the window size, and is more sensitive to large variations and peaks in the observed data. These results validate the choice of the Kriging method to predict CPU consumption in a dynamic context. Furthermore, we propose historical data-based adjustment, which improves the efficiency of the spatial locality-based Kriging prediction and reduces the under-estimation caused by sharp variations in resource usage.

2.9.2.2 Prediction and Adjustment

First, we defined a set of alternative scenarios, namely: prediction with fixed-size sliding window and fixed number of predicted data: $(n_i, m_i) = (10,5)$, and without adjustment; prediction with variable number of predicted data: $(n_i, m_i) = (10, [7,5])$, adjustment and standard deviation-based padding (strategy 3 defined in the Section 2.9.2.3 padding strategies), and prediction with the sliding window size and the number of predicted data selected dynamically by GA (Algorithm 2.4), adjustment and standard deviation-based padding (strategy 3). The sliding window and predicted data size values (n_i, m_i) for scenarios 1 and 2 were selected among initial GA population pairs that had provided better prediction performance in prior Kriging prediction tests. Next, we experimented with several types of variations/fluctuations of CPU usage with a small period (5 sec) or a large period (5 min) of data collection (e.g., Conf1 and Dset1, respectively) to evaluate our approach to dealing with various workload situations.

Table 2.7 details the characteristics of our scenarios. In all the scenarios, the prediction is dynamic and adaptive to the workload profile. However, the sliding window size and the

number of predicted data are adaptive and dynamic only in the third scenario, which represents our approach. Furthermore, the adjustment and the padding are dynamic and adaptive in the second and the third scenarios. By defining and testing these scenarios, we aim to compare the impact of each proposed technique on the CPU consumption prediction accuracy and the cost in terms of under-estimation and over-estimation.

Table 2.7 Characteristics of prediction scenarios

Scenario	Strategy	Adaptive	Dynamic
Scenario1 Without Adjustment, Padding and GA	Sliding window size, Number of predicted data	X	X
	Prediction - Kriging	✓	✓
	Adjustment and Padding	Does Not Apply	
Scenario2 Adjustment and Padding without GA	Sliding window size, Number of predicted data	✓	X
	Prediction - Kriging	✓	✓
	Adjustment and Padding	✓	✓
Scenario3 Adjustment, Padding and GA	Sliding window size, Number of predicted data	✓	✓
	Prediction - Kriging	✓	✓
	Adjustment and Padding	✓	✓

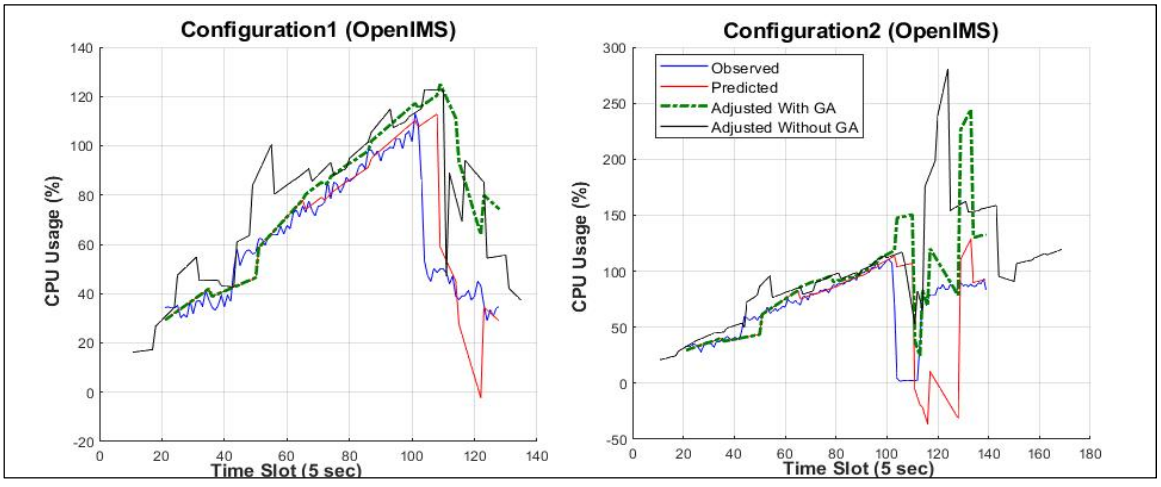

Figure 2.6 Prediction and adjustment of CPU consumption in cases of Configuration 1 and 2 (OpenIMS)

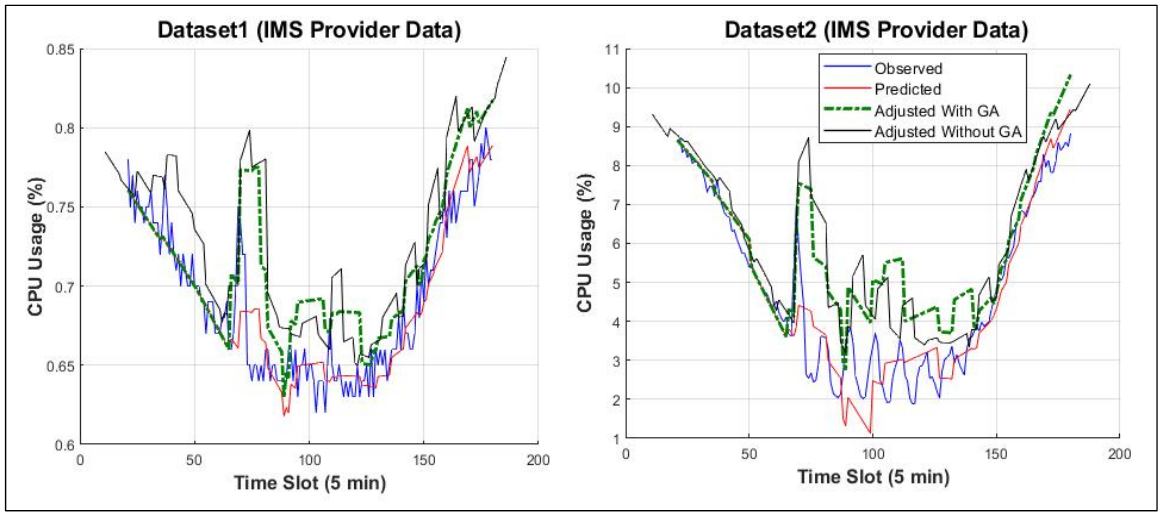

Figure 2.7 Prediction and adjustment of CPU consumption
in cases of Datasets 1 and 2 (IMS)

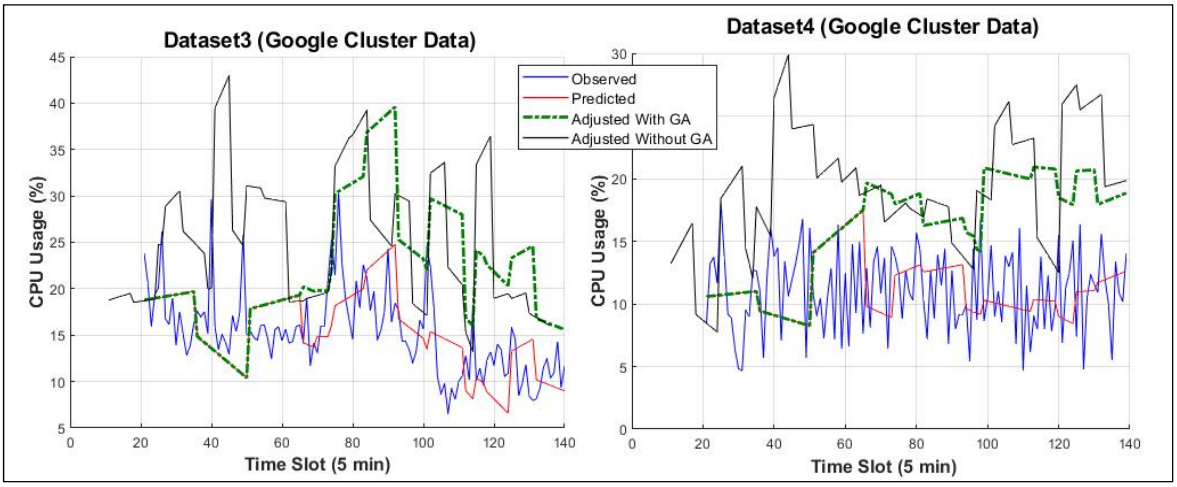

Figure 2.8 Prediction and adjustment of CPU consumption
in cases of Datasets 3 and 4 (Google cluster data)

Figure 2.6, Figure 2.7 and Figure 2.8 present the CPU consumption prediction results for the defined scenarios and various systems and workload profiles. During the initialization phase, several training and prediction steps using the Kriging method are performed (without adjustment) until sufficient data for applying adjustment and optimization are collected. Therefore, we observe under-estimation cases before the 60 time slot (Figures 2.6, 2.7 and 2.8).

Mainly, the results show that the prediction using the Kriging method can follow the CPU usage trend, and can adapt to changes for all considered configurations and datasets. However, it is less effective for large and unexpected variations (e.g., Configuration2, see Figure 2.6) or many fluctuations (e.g., Dataset4, see Figure 2.8) occur, which cause long and significant under-estimations and SLA violations. Therefore, we propose the adjustment of the prediction and the standard deviation-based padding (strategy 3 outperforms in almost all scenarios and datasets: see Section 2.9.2.3 Padding strategies) to reduce long and/or frequent under-estimations. The adjusted prediction and variable padding results show a clear improvement in terms of significant under-estimation reduction.

However, we observe some cases of large over-estimation, for instance: Dataset3 and Dataset4 (Figure 2.8, plot: adjusted without GA). These over-estimations arise mainly when the variation and the magnitude of the CPU consumption in the current prediction phase are different from the previous one. It is mainly due to the adjustment and the padding values computed using the results of previous prediction steps. Further, we notice from several experiments that the prediction performance is influenced by the size of the sliding window and the number of predicted data. Therefore, we use GA to dynamically provide the size of the sliding window and the number of predicted data minimizing under-estimation and over-estimation at each prediction step. The results of the adjusted prediction with the use of GA (see Figures 2.6, 2.7 and 2.8 plots adjusted with GA) mostly show a decrease in over-estimation, while under-estimation decreases as well (e.g., see Figure 2.8: Dataset3 and Dataset4) or is close to the results of the adjusted prediction without GA (e.g., see Figure 2.6: Configuration1, Configuration2). Therefore, we conclude that the accuracy and the adaptability of our algorithm are improved significantly thanks to the prediction and the adjustment techniques combined with GA.

To evaluate and quantify the efficiency of our algorithm, we compute the probability of under-estimation, the mean of over-estimations, the mean of over-estimations for threshold-based provisioning, and the mean time between under-estimations (*MTBUE*) for both predicted and

adjusted data. Figure 2.9 summarizes the evaluation metrics for the defined scenarios, configurations and datasets.

As shown in Figure 2.9.c, the prediction without any adjustment is characterized by a large probability of under-estimation (between 0.44 and 0.54) (Figure 2.9.c1), a mean of over-estimation under 10% for all configurations and datasets (Figure 2.9.c2), and a short MTBUE: less than 17 seconds for all configurations and less than 21 min for all datasets (Figure 2.9.c3). These results reveal the limitations of using the prediction without adjustment specifically in the presence of fluctuating workloads.

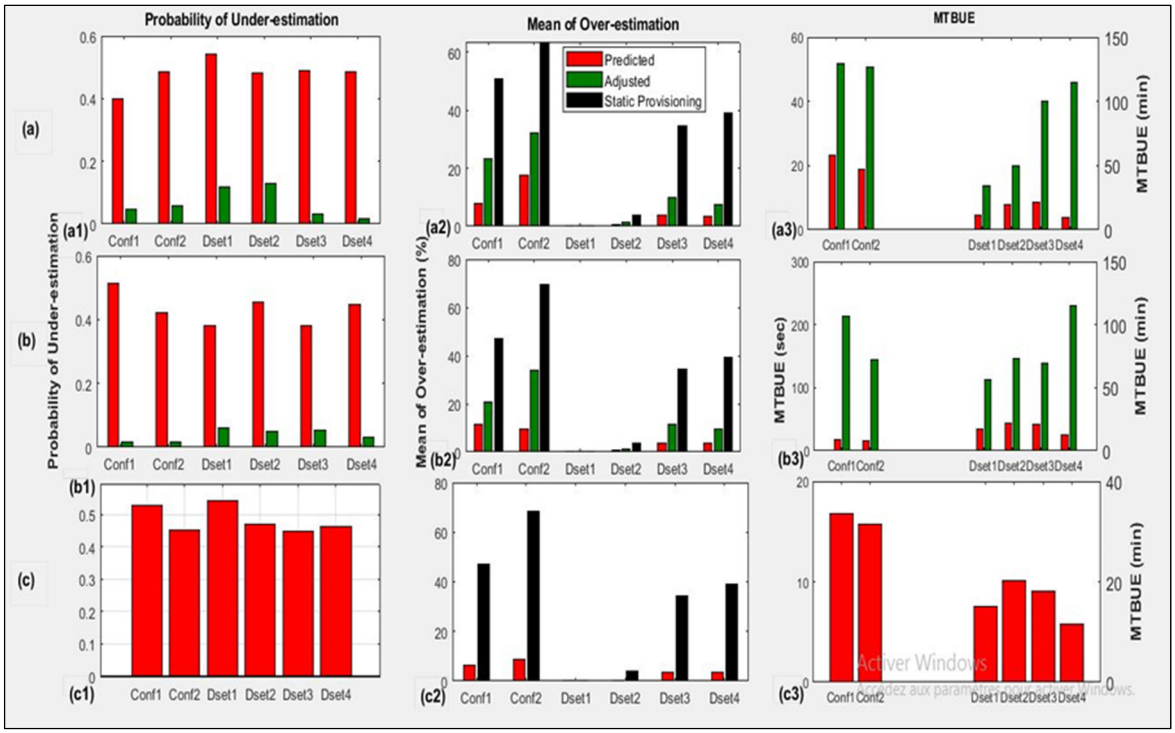

Figure 2.9 Results of Evaluation Metrics in case of: (a) Prediction (dynamic sliding window), Adjustment and GA, (b) Prediction with fixed-size sliding window, Adjustment without GA, (c) Prediction with fixed-size sliding window and fixed number of predicted data

A comparison of the results presented in Figure 2.9.a1, Figure 2.9.b1 and Figure 2.9.c1 shows that the proposed prediction with adjustment and padding (strategy 3) minimizes the under-estimation for all configurations and datasets remarkably. Indeed, the prediction adjustment

allows reducing the under-estimation by 86% on average as compared to prediction without adjustment. For instance, the probability of under-estimation decreases from 0.39 to 0.04 for Configuration1, and from 0.48 to 0.01 for Dataset4 (see Figure 2.9.a1). Moreover, our approach results in a significant increase in the mean time between under-estimations (MTBUE). For example, the MTBUE increases from 20 min to 50 min and from 10 min to 115 min in the case of Dataset2 and Dataset4, respectively (see Figure 2.9.a3).

Regarding resource wastage due to over-estimation, our algorithm significantly improves the efficient consumption of CPU resources in comparison with threshold-based provisioning (static provisioning) for both prediction and adjustment, for all configurations and datasets. Actually, our approach is able to reduce over-estimation by 67% on average, as compared to the threshold-based provisioning (see Figure 2.9.a2 and Figure 2.9.b2). For instance, the mean of over-estimation of CPU consumption decreases from 64% to 32% and from 0.31% to 0.03% for Configuration2 and Dataset1, respectively (see Figure 2.9.a2).

The main improvements provided by GA to the proposed algorithm are the dynamic selection of the sliding window size and the number of predicted data, as well as the flexibility and the adaptability to changes in the workload while minimizing SLA violations and resource wastage. As shown in Figure 2.9.a and Figure 2.9.b, the probability of under-estimation decreases (see Figure 2.9.a1 and Figure 2.9.b1), and the MTBUE increases when using GA in both Dataset3 and Dataset4 (see Figure 2.9.a3 and Figure 2.9.b3). Further, the scenario with GA improves the over-estimation results (see Figure 2.9.a2 and Figure 2.9.b2), in almost all configurations and datasets, except for Configuration1 and Dataset2. For instance, the usage of GA enables decreasing the probability of under-estimation and the mean of over-estimation from 0.05 to 0.03 (see Figure 2.9.a1 and Figure 2.9.b1) and from 12% to 10% (see Figure 2.9.a2 and Figure 2.9.b2), respectively, for Dataset3. In contrast, the scenario without GA gives better results for Configuration1, Configuration2, Dataset1 and Dataset2. However, the probability of under-estimation, as well as the mean of over-estimation, remain close for the two scenarios (adjustment without vs. with GA). This difference in performance between scenarios 2 and 3 (without GA, with GA, respectively) may be explained by the use of a small

and prefixed sliding window and predicted data sizes (10, [7,5]) in scenario 2, which allows a quick adaptation to recent variations. However, scenario 2 loses in flexibility and adaptability of prediction, and is less accurate in the case of a fluctuating workload (e.g., Figure 2.8, Dataset3 and Dataset 4). Therefore, the challenge is to find the optimal trade-off between the cost of SLA violations and the cost of resource wastage (minimizing under-estimation and over-estimation), while ensuring adaptability and flexibility for the prediction algorithm.

2.9.2.3 Padding strategies

First, we computed the padding value dynamically based on the prediction error and the ratio *r* between the adjusted data and the observed data (see Equation 2.5). We obtained good results in terms of under-estimation probability, but the mean of over-estimation was observed to be significantly higher when the workload tended to fluctuate.

Then, we tested standard-deviation-based (std) padding with different strategies by considering:

- strategy 1 (std1): the std of observed data in the previous prediction step.
- strategy 2 (std2): the mean of the previous std that were computed in the case of under-estimation greater than 10%.
- strategy 3 (std3): the mean of previous std that were computed in the case of under-estimation greater than 10%. If the current std value (current under-estimation >10%) is greater than twice the previous std value ($\sigma_{current} > 2\sigma_{previous}$), it is excluded from the mean std estimation.

The results of the std-based padding strategies show an improvement in terms of reducing over-estimation. However, the selected strategy, namely, strategy 3 (see Algorithm 2.3), outperforms in almost all scenarios and for all datasets. For instance, the mean of over-estimations of adjusted data in the case of Configuration2 with ratio-based padding is about 41.88, whereas the std-based padding reduced the over-estimation mean to 31.26 (std3). Table 2.8 presents the evaluation metrics in each padding strategy using Configuration2 data.

Table 2.8 Results of evaluation metrics of various padding strategies for configuration2

Padding Strategy	Probability of under-estimation	Mean of over-estimation (%)	Mean of over-estimation Static provisioning (%)	MTBUE(sec)
ratio	**0.05**	41.88	65.3	**54.16**
std1	0.07	31.55	64.48	49.375
std2	0.058	32.28	**63.61**	50.71
std3	0.06	**31.26**	64.48	50

2.10 Conclusion and future work

This work presents a generic, dynamic and multi-step ahead prediction of resource demand in virtualized systems. Based on a time series and machine learning method, our proposed algorithm is able to provide real-time prediction of resource needs without any prior knowledge or assumptions on the system or its internal behavior. When unexpected workload fluctuations occur, the proposed algorithm is capable of adapting to these changes within a relatively short delay (e.g. on average, 40 seconds in Configuration 2). Our algorithm also includes a dynamic adjustment based on the estimation of prediction error probability, and padding strategies to minimize SLA violations and reduce over-estimation. Furthermore, the proposed algorithm is able to dynamically generate the size of the sliding window and the number of predicted data, bringing flexibility to the prediction and improving its performance. Thorough experiments were conducted using various virtualized systems and different workload profiles. The results show that our algorithm is able to reduce the under-estimation average by 86% as compared to prediction without adjustment. Further, our algorithm decreases the over-estimation average by 67% against threshold-based provisioning. For future work, additional experiments and analyses using datasets from different types of systems and applications would be valuable to reinforce the general characteristics of our algorithm. Additionally, further investigations and evaluations should be conducted in order to improve the adjustment delay, the trade-off between under-estimation and over-estimation, as well as the trade-off between historical data size and computational efficiency. Moreover, tests and prediction performance comparison

using various machines learning (e.g. hybrid approach using Kriging and SVR), as well as investigations of abnormal behavior will be conducted.

Acknowledgment

This work has been supported in part by Natural Science and Engineering Research Council of Canada (NSERC), in part by Ericsson Canada and in part by Rogers Communication Canada.

CHAPITRE 3

ABNORMAL BEHAVIOR DETECTION USING RESOURCE LEVEL TO SERVICE LEVEL METRICS MAPPING IN VIRTUALIZED SYSTEMS

Souhila Benmakrelouf [a], Cédric St-Onge [b], Nadjia Kara [c], Hanine Tout [d], Claes Edstrom [e], Yves Lemieux [f]

[a, b, c, d] Department of software engineering and IT, École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[e, f] Ericsson Canada,
8275 route Transcanadienne, Saint-Laurent, Québec, Canada, H4S 0B6

Published in "Journal of Future Generation Computer Systems",
July 2019

3.1 Abstract

Recent years have seen a pronounced growth in research interest in resource management of virtualized systems. In that context, several techniques have been proposed. However, this task is complicated by the heterogeneity and virtualization of resources and the scalability and high-performance service requirements of Cloud Computing. Moreover, unpredictable situations such as fluctuations in resource demand or abnormal changes in systems behavior due, for instance, to emergency situations or special events, cause critical service performance degradation and costly SLA violations. Existing approaches are not generic, and are based mainly on excessive allocation of resources, a prospect which is not efficient, and involves resource loss. Further, they propose monitoring systems without a validation, using correspondence between service levels and resource levels and pattern detection. To overcome these limitations, in this work, we propose an effective, dynamic and real-time detection of abnormal behavior. Our approach is based on outlier detection techniques and employs a mapping between service level metrics and resource level metrics, as well as periodicity detection and effective system state analysis and notification. Experiments conducted on several types of workloads showed that our algorithm is capable of detecting unusual changes

in both service level and resource level metrics. It is also able to map detected changes and provide corresponding notifications with an accuracy ranging from 92% to 100%.

Keywords: divergence, mapping, outlier detection, periodicity, resource level metric, service level metric.

3.2 Introduction

Virtualization is one of the crucial technologies that strongly influence the delivery of next-generation network services. It enables scalability, optimized sharing and flexible management of resources, as well as reducing cost in data centers. To take advantage of this technology, virtualized systems must dynamically and continually be provisioned with resources according to their workload demands. However, the resource management of virtualized systems is still a challenging task because of the complexity of such systems, their dynamic and heterogeneous environments, and their fluctuating resource demands over time (Shyam & Manvi, 2016), particularly in a context where resource utilization has to be optimized and SLA violations avoided.

Effectively and accurately predicting resource demand is essential for anticipating resource needs and managing their utilization. Nevertheless, the main drawback of resource prediction is that it involves workload fluctuations and significant and unusual changes in systems behavior. In such situations, resource prediction and resource planning are unable to anticipate and provide required resource quantities, which consequently affects the quality of service and causes SLA violations or resource loss in the case of an increase or decrease in system demand, respectively.

Furthermore, abnormal and sudden workload changes owing, for instance, to emergency situations, special events, as well as failures in system components or monitoring tools, are just some of the resource management challenges in virtualized systems. Such unpredictability causes critical service performance degradation and costly SLA violations. An excessive

allocation of resources as a solution to this problem is also not efficient, and leads to resource loss.

To detect abnormal behaviors, patterns and significant system changes, we propose a novel solution in this work; the solution is generic enough to be applied to different virtualized systems or applications, can detect and evaluate abnormal system changes (service and resource levels) dynamically and in real-time, analyze and detect periodicity and large variations, map high-level service metrics to low-level resource metrics, and generate notifications that specify a decision and its priority in resource planning. Our approach provides an algorithm for the dynamic and accurate detection of abnormal systems behavior by developing and leveraging different techniques. The proposed algorithm mainly employs two techniques, namely, Kullback-Leibler divergence and Modified Z-Score, to detect unusual workload and resource utilization variations. Further, our algorithm is able to analyze collected data and to detect patterns and large variations using a novel method called Weighted digital print deviation. As well, the algorithm analyzes the results of metrics mapping and provides the type and the priority of workload changes (e.g., high priority notification when opposite changes occur), and sends notifications to the resource planning component regarding unusual workload change, its repetition and its priority.

Thorough experimentation has been conducted to study the efficiency and performance of the proposed algorithm and techniques with different systems and workloads. The main contributions of this work can be summarized as follows:

1. Novel algorithm for dynamic and real-time detection of abnormal variations and mapping of service level metrics to resource level metrics (e.g., throughput and CPU usage) in a virtualized system, without any prior knowledge or assumptions on its internal behavior,
2. Dynamic and real-time workload analysis to detect unusual variations in the system behavior based on combined divergence and outlier detection techniques,
3. Dynamic periodicity detection algorithm that can detect patterns, as well as significant variations characterizing a possible unusual behavior of the system,

4. Dynamic and real-time evaluation of the system state and generation of resource adaptation notifications or warnings,

The rest of the paper is organized as follows. First, we review the state of the art related to unusual behavior detection using outlier detection techniques and the mapping of resource and service level metrics in the context of resource management in virtualized systems. Next, we present the approach we propose to detect unusual variations, and we explain the algorithm, the methods and the techniques we used. Then, we evaluate the performance of our algorithm. Finally, we analyze and discuss the main results and conclude the article.

3.3 Related work

Over the past few years, resource management of virtualized systems has grown in importance, with a wide scope being presented for the development of adequate efficient management techniques. In what follows, we study relevant existing techniques in this context and we classify them in Table 3.1, based on key features needed for efficient detection of abnormal behavior in workloads based on outlier detection techniques and on the mapping of resource and service level metrics in virtualized systems.

3.3.1 Outlier detection

Outliers are unusual observations in the data which do not conform to the expected behavior. They indicate significant but rare events, and they can prompt critical actions to be taken in different application domains. Outlier analysis (Aggarwal, 2013) involves automatically discovering interesting and rare misbehaviors from datasets.

While clustering, which is an unsupervised learning method, is a powerful method for identifying normal and anomalous data, the authors in (Ahmed, Mahmood, & Islam, 2016) surveyed different clustering-based techniques for fraud detection. The work in (Ahmed, Naser Mahmood, & Hu, 2016) presented a survey on network anomaly detection techniques and

categorized them into four classes of classification-based, statistical, information theory and clustering-based techniques. Yang et al. (Z. Yang, Meratnia, & Havinga, 2010) also surveyed different outlier detection techniques for wireless sensor networks, which they categorized into statistical-based, nearest neighbor-based, clustering-based, classification-based, and spectral decomposition-based approaches.

The Kullback-Leibler divergence (KLD) derived from information theory-based techniques, has been used for abnormality detection and pattern recognition in different areas. An incipient fault detection method that does not require any a priori knowledge on the signal distribution or the changed parameters was proposed in (Youssef et al., 2016). This method is based on the analysis of the Kullback-Leibler divergence of probability distribution functions.

Takeuchi et al. (Takeuchi & Yamanishi, 2006) proposed a framework for detecting outliers and change points from time series. Outlier detection is related to fraud detection, rare event discovery, etc., while change point detection is related to event/trend change, activity monitoring, etc. The proposed framework considers an average of scores over a window of a fixed length and a sliding window in order to provide a new time series consisting of moving-averaged scores. Change point detection is then conducted by detecting outliers in that time series, and similarity is measured using KLD.

Li et al. (Li & Wang, 2012) also applied the KLD technique to detect anomalous data values in wireless sensor networks. The network is divided into clusters. In each cluster, the sensors remain physically close to each other and sense similar values. After clusters are formed, outlier detection is performed using KLD. In particular, the divergence between the median value dataset and that of a cluster node is calculated using the KL divergence metric, with outliers being identified as those with a metric below a predefined threshold. In (Salem et al., 2012), Salem et al. propose a sequential approach to identifying anomalies in network traffic. They use Jensen-Shannon Divergence (JSD) over a sketch data structure. It measures the difference between two sets of probability values to detect the deviation between the sets in normal and abnormal conditions.

(Jing et al., 2013) studied the management of resources according to fluctuating loads in SaaS (Software as a Service) applications, while meeting users' expectations of Quality-of-Service (QoS). Jiang et al. proposed a cloud resource auto-scaling scheme, at the virtual machine (VM) level, for Web application providers to achieve both true elasticity and cost-effectiveness within a pay-per-use cloud business model. Additionally, they evaluated the seasonal character of collected data by applying KLD and Symmetrizing KLD to measure the difference between two distributions (e.g., hourly distribution: 24-length vector of hourly number of requests). The results showed that analyzed data are highly seasonal, which ensures the relevance of historical data in predicting the number of requests.

3.3.2 Mapping service level to resource level metrics

Service providers must be able to monitor their infrastructure resource metrics in order to meet SLA requirements. However, in traditional systems, there is a clear gap between monitored metrics, which are usually low-level metrics (resource level), and SLA agreements, which are high-level metrics (service level) (Emeakaroha, Brandic, Maurer, & Dustdar, 2010). Further, some works proposed a mapping of virtual resources to candidate substrate resources (Gil Herrera & Botero, 2016) or a mapping the service and SLA into network function level and infrastructure level (Zhou, Li, Chen, & Zhang, 2016).

Emeakaroha et al.(Emeakaroha et al., 2010) proposed a framework for mapping low-level resource metrics to high-level SLAs. The framework detects future SLA violation threats and puts out a notification to act based on: run-time monitoring of services and resources metrics, and mapping one-to-one from metrics or using predefined rules, defined by the service provider, in order to link resource metrics to SLA parameters (e.g., Availability = downtime/uptime), and predefined threat thresholds. Relying on predefined rules and threat thresholds, this proposition lacks generality and cannot be applicable in diverse applications and systems.

In (Nguyen, Shen, Gu, Subbiah, & Wilkes, 2013), the authors propose an elastic distributed resource scaling system for IaaS Cloud infrastructure, which includes medium-term resource demand predictions using Wavelets for achieving enough time to scale up the server pool, and pre-copy live cloning to replicate running VMs. They also use online profiling and polynomial curve fitting (polynomials with different orders) to provide a black box performance model of the application's SLO violation rate for a given resource pressure (i.e., ratio of the total resource demand to the total resource allocation for the server pool). Because the resource pressure model is application-specific, a set of models is maintained dynamically, and the best model is selected for the current workload (using the least square error). The performance of such a proposition may be significantly affected in the case of a fluctuant workload due to the generation and adjustment of new models (10 to 20 minutes to derive a new resource pressure model from scratch using the online profiling scheme).

3.3.3 Periodicity detection algorithms

Research on periodicity mining has grown significantly, with a focus on various areas (Faraz Rasheed & Alhajj, 2008), (F. Rasheed & Alhajj, 2014), (Ahdesmaki, Lahdesmaki, Pearson, Huttunen, & Yli-Harja, 2005) ,(Elfeky, Aref, & Elmagarmid, 2005) (Chanda, Saha, Nishi, Samiullah, & Ahmed, 2015) (Chanda, Ahmed, Samiullah, & Leung, 2017). The approach proposed in (Ahdesmaki et al., 2005) aims to find periodicity in biological time series. Based on spectral analysis and different hypothesis testing schemes, a decision model is proposed to find periodicities. Contrarily to the frequency spectrum analysis, this work detects periodicity based on discrete workload data. However, the method used for estimating the cell cycle length/frequency motivates the conversion of discrete data into continuous splines in order to get continuous redesigned values.

Other periodicity detection approaches (Faraz Rasheed & Alhajj, 2008), (F. Rasheed & Alhajj, 2014), (Elfeky et al., 2005), (Chanda et al., 2015), (Chanda et al., 2017), (Nishi, Ahmed, Samiullah, & Jeong, 2013) have been applied to study stock price variation, network traffic density, gene expressions and many other aspects. These approaches discretize time series data

into symbols and apply symbol periodicity in the early stage of pattern detections. Algorithms proposed in (Faraz Rasheed & Alhajj, 2008), (F. Rasheed & Alhajj, 2014) and (Nishi et al., 2013) are based on suffix tree mechanisms to classify periodic patterns in symbols and detect partial periodicity in time series. Occurrences of all repeated patterns in data are captured by building the suffix tree, and then the periodic strength of these patterns are calculated by checking the repetitions of these periodicities.

With the ability to mine flexible patterns, the solution presented in (F. Rasheed & Alhajj, 2014) improves the work in (Faraz Rasheed & Alhajj, 2008), which only detects fixed-length periodic patterns. Giving more significance to less frequent yet periodic patterns, (F. Rasheed & Alhajj, 2014) can detect the periodicity of outlier patterns.

Similarly, the algorithm proposed in (Nishi et al., 2013) leverages suffix tree-based periodicity detection aimed at helping the user generate a variety of patterns by skipping intermediate events while discovering the periodicity of patterns within a database. While such an approach is efficient in situations demanding user interaction to define patterns, another strategy is required for automatic workload periodicity detection that necessitates minimal user intervention.

Other suffix tree-based approaches are also used for symbolic pattern detection (Chanda et al., 2015) (Chanda et al., 2017). While varying the starting positions, these approaches generate periodic patterns with and without skipping unimportant intermediate events in time series databases. By handling different starting positions simultaneously, the algorithm in (Chanda et al., 2015) stimulates flexibility among events in the mined patterns and enables interactive and on-the-go tuning of period values. On the other hand, the work in (Chanda et al., 2017) can detect all single, partial and full-weighted periodic patterns in a single run. Thus, faster detection of the most important patterns is achieved.

Although solutions discretizing data from datasets into symbols are efficient, one major shortcoming with them is their inability to detect and classify cycles based on the shapes and

amplitudes of the latter. In other words, these are cycles with certain shapes that may reoccur in the dataset with higher or lower values, or even those with similar amplitudes, but different shapes.

3.3.4 Workload pattern recognition on prefix transreversal

Pattern recognition in workloads has also been studied, and offer the capacity to detect anomalies such as CPU-intensive loops, memory leaks, disk I/O errors and network anomalies. The work in (T. Wang, Wei, Zhang, Zhong, & Huang, 2014) presents a mechanism for detecting faulty Web applications and locating fishy activities suspected of performing anomalous actions. The periodicity detection approach used in the work is based on workload vectors that characterize dynamic workloads. Such an approach is useful in defining combinations of amplitude, shape and length in a bid to identify anomalies in datasets.

Other interesting research is centered around prefix transreversal algorithms (Dutta, Hasan, & Rahman, 2013) (Khaledur Rahman & Sohel Rahman, 2015) (Dias & Dias, 2015), which are used by molecular biologists and bioinformaticians as a means to trace evolutionary distances between pairs of species. Basically, a prefix transreversal problem can be formulated as follows: given two permutations, find a shortest sequence of rearrangement operations that transforms one given permutation into the other one. This paper presents a novel technique that combines transreversal computations with binary strings to provide a strong and flexible pattern detection model capable of detecting patterns regardless of their shape, length and amplitude.

3.3.5 Proposed approach

Our approach differs in many aspects from what obtains in the studies presented in the State of the art section, and provides a generic and dynamic solution for detecting abnormal behavior in virtualized systems. While existing approaches are application-specific and/or use predefined parameters and rules, ours provides a generic solution that is capable of detecting unusual variations in various workload and metrics types (e.g., CPU, throughput, memory,

latency) without any prior knowledge of the system or any assumption on its behavior or load profile. Furthermore, our proposed algorithm provides a mapping between resource level and service level metrics to verify system behavior at both levels, and analyzes occurrences of abnormal variations and the periodicity of detected patterns. It also generates resource adaptation notifications and warnings based on detection, mapping and analysis process results. Moreover, our advanced periodicity algorithm follows a different approach, which leverages prefix transreversal operations and a novel “digital print” extraction mechanism to screen the dataset in order to detect deviation patterns, and offers a general purpose and generic periodicity detection algorithm that fits different types of telecom and IT workloads. These two types of workloads are different from each other. Usually, IT CPU workloads show sharper variations in short bursts, while telecom workloads are flatter, continuous loads under normal customer demand. Contrarily to existing techniques, the approach adopted here automates the whole process while allowing the detection of patterns without being limited to any amplitude, shape or length. It also offers the ability to identify similar workload patterns in a dataset as well as significant variations, which facilitates improving load profiles for workload modeling.

3.4 Approach overview

Unanticipated abnormal behavior in system characteristics can potentially lead to performance degradation or service failure, and might even end in costly SLA violations. The events responsible for generating changes in workload can either be internal to the application, or due to an exogenous influence. Their occurrence can be well-known in advance, (for example, special events) or completely unexpected events, such as emergency situations and failures in system components or monitoring tools. Detecting abnormal behavior is complicated by the fact that there is no coherent definition of such variations. More specifically, what appears to be an abnormal change for one application may be considered as normal for another one, which makes it harder to any attempt to characterize spikes. Further, when a system behavior analysis is done at the service level independently from the resource level, the evaluation of the system state is incomplete, and results in wrong decisions. In this work, we propose a generic, dynamic, and accurate detection of system abnormal behavior using outlier detection

techniques, a mapping of resource level to service level metrics and periodicity detection. Our aim is to minimize the reaction time of the system when unusual changes in workload occur by providing a continuous, fast and precise evaluation of the system state and the action to be undertaken. Ultimately, we want to minimize possible SLA violations and loss of resources, without any prior knowledge of the system or any assumption on its behavior or load profile. To that end, we propose a novel algorithm for detecting abnormal behavior in systems as well as unpredictable workload variations.

Figure 3.1 presents the main components of the proposed abnormal behavior detection algorithm. It includes two main techniques. The first technique leverages the Kullback-Leibler divergence to analyze the variation between observed data in each sliding window. The second technique considers the Modified Z-Score to detect significant changes in workload (detection of outliers) and provide the position and the type of change (workload increase or decrease). Afterward, the results of outlier detection techniques for both service level and resource level metrics are mapped in order to provide the type of workload changes (e.g., metrics with the same or opposite changes) and their significance (e.g., high, medium, low priority). Additionally, the periodicity in terms of cycles and large variations of workload is examined, and as the change recurrence in a sliding window is identified to track the workload change tendency. Finally, we provide a notification mechanism to alert (e.g., trigger resource adaptation or warnings) the resource planning component regarding an unusual workload change, its repetition and its priority. The latter analyzes the received notifications, the current and the predicted workloads (Resource Prediction), and estimates the amount of resources needed and the available time to deploy them accordingly. While this section gives an overview of the proposed solution, details are only provided in the following sections. It should be noted that the resource prediction and planning components are beyond the scope of this work. Table 3.2 describes the notations and the symbols used in the rest of the paper.

Table 3.1 Taxonomy of Abnormal behavior detection approaches

Reference	Criteria							
	Target	**Genericity**	**Real-time**	**Detecting abnormal behavior technique**	**Metric**	**Periodicity**	**Number of occurrences**	**Notification**
(Youssef et al., 2016)	Fault detection on the signal distribution	Yes	Offline	KLD of probability distribution functions.	Experimental data	No	No	Yes
(Takeuchi & Yamanishi, 2006)	Detection of outliers and change points	No	Offline	Moving-averaged scores Measure of similarity using KLD	Frequency of connection requirements	No	No	Yes
(Li & Wang, 2012)	Detection of anomalous data in wireless sensor networks	No (predefined threshold)	Offline	KLD	Data collected from sensors	No	No	No
(Salem et al., 2012)	Identification of anomalies in network traffic	No	Run-time monitoring	Jensen-Shannon Divergence (JSD)	Network traces	No	No	Yes
(Jing et al., 2013)	Cloud resource auto-scaling scheme Evaluation of seasonal character of collected data	Yes	Offline	KLD and Symmetrizing KLD	Number of requests	Difference between 2 time periods	No	No
(Emeakaroha et al., 2010)	Mapping of low-level resource metric to high-level SLA parameters	No (specific to defined SLA and resources)	Run-time monitoring	Predefined mapping rules and thresholds	SLA parameters	A set of models is maintained dynamically	No	Yes
(Nguyen et al., 2013)	Elastic distributed resource scaling system including online profiling	No (models maintained dynamically)	Run-time monitoring	No	SLO and resource pressure	No	No	Yes
Our approach	Abnormal behavior detection	Generic	Real-time	KLD, Modified Z-Score	CPU usage, memory usage, latency, throughput	Cycles and large variations	Yes	Yes

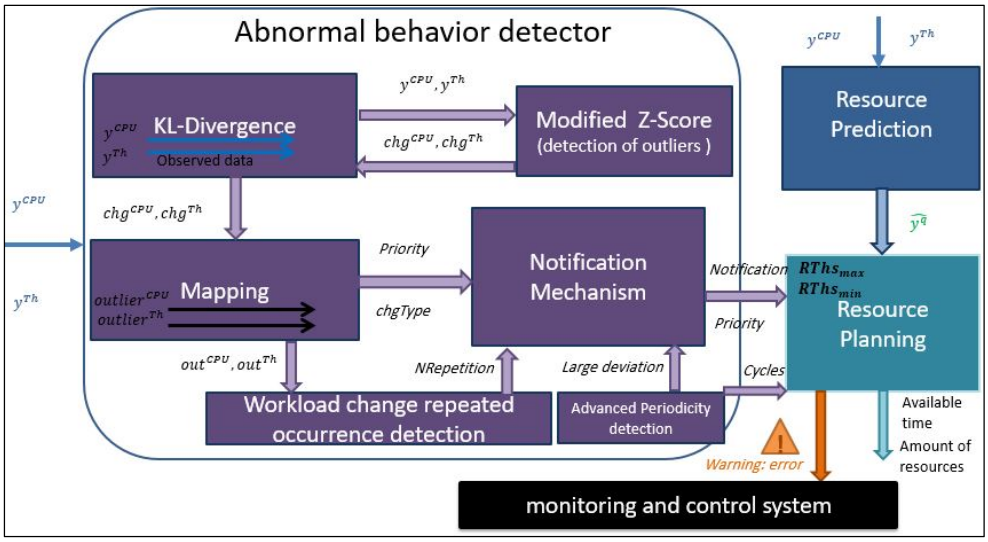

Figure 3.1 Abnormal behavior detector components

Algorithm 3.1 presents the proposed approach for abnormal behavior detection. It starts by initializing arrays of detected outliers and their corresponding types of change, while the notification type and priority are set to zero. It also defines the categories of change types, for instance, warning types, increase types and decrease types (Line 1 to Line 4). At each time slot, the collected resource and service level metrics data are analyzed using the Kullback-Leibler divergence and Modified Z-Score to detect possible outliers that reflect significant changes in system behavior (Algorithm 3.2, Line 6) as well as their types (i.e., increase, decrease, and steady). The results of the outlier detecting algorithm, for both metrics, are saved (Line 7) and mapped by adding the type of change of each resource level metric value to its corresponding service level metric value (Line 8). Afterwards, the mapped changes obtained and the observed data are analyzed to provide the type and priority of each mapped change (Algorithm 3.3, Line 9), as well as the number of recurrences of equal and successive changes (Lines 10-12). Next, a search for cycles and large deviations in both metrics is performed (Algorithm 3.4, Lines 14 and 15) based on the data collection history at each sliding window (Line 13) to consolidate decision making in the notification mechanism component. According to the type, priority and recurrence of changes, as well as detected cycles and large deviations, a notification of a proposed decision and its priority are generated (Algorithm 3.5, Line 16)

and sent to the resource planning component (Line 17). The proposed mapping process runs constantly with continuous system monitoring and data collection to detect changes, analyze their types and send relevant adequate notifications.

Algorithm 3.1 Complexity: The time complexity of the proposed algorithm for abnormal behavior detection depends essentially on four parts: the time taken by the outlier detection techniques, the time complexity for identifying the type, the priority and the number of occurrences of mapped metrics changes, the time complexity for periodicity detection, and the time complexity for determining the type and the priority of notifications. First, the time complexity of the parameters' initialization (e.g., outliers, types of change, type and priority of notification) is *O(1)* (Lines 1 to 4). Next, the detection of outliers using Kullback-Leibler divergence and Modified Z-Score (Line 6, Algorithm 3.2) is performed with a time complexity of $O(n^2)$, where *n* is the window size of the collected data. Further, change types are saved and detected outliers mapped with a time complexity of *O(1)* (Lines 7 and 8). Then, the type and the priority of each mapped change are deduced by Algorithm 3.3 (Line 9), with a time complexity of *O(1)*. For each mapped change type, the number of recurrences of equal and successive changes is computed (Lines 10 to 12), with a time complexity of $O(nk)$, where *k* is the number of repeated changes having $k < n$. Next, the data collection history is gathered (Line 13) with a time complexity of *O(1),* while the detection of cycles and large variations is performed (Lines 14 and 15) with a time complexity of $O(h.l)$, where *h* is the size of the data collection history, and *l* is the length of the binary string being evaluated. Finally, the notification type and priority are determined and sent to the resource planning, with a time complexity of $O(Tn^2)$, where *T* is the change type number. Consequently, the time complexity of Algorithm 3.1 is $O(1)+O(n^2) + O(1)+O(1)+O(1)+ O(nk) +O(1) + O(h.l) + O(T.n^2)$, which is equivalent to $O(Tn^2)$.

Algorithm 3.1 Abnormal Behavior Detection

Abnormal Behavior Detection (y_i^{CPU}, y_i^{Th}, *m, DiverThreshold*)

Input: (y_i^{CPU}, y_i^{Th}, *m, DiverThreshold*)
Output: Notification, Priority

1: Initialize arrays of outliers: CPUOutliers:=zero[], ThOutliers:=zero[]
2: Initialize types of metrics changes: varType = [1,2,3,4,5,6,7,8]
3: Define categories of change types: warningType:=[3,4,6,8], increaseType:=[1,7], decreaseType:=[2,5]
4: Initialize historic data, notification type and its priority:
Hy^{CPU}=[], Hy^{Th}=[], Type:=zero[], Priority:=zero[]
5: ***for*** each collected data window
6: Detect divergence and outliers in collected data:
(div_{CPU}, div_{Th}, chg_{CPU}, chg_{Th}) = **Algorithm 3.2** (y_i^{CPU}, y_i^{Th}, *m, DiverThreshold*)
7: Save change types of detected outliers:
CPUOutliers:= chg_{CPU}, ThOutliers:= chg_{Th}
8: Map detected outliers in both metrics:
SumOutliers:= CPUOutliers + ThOutliers
9: Determine the type and the priority of each mapped change:
(Type, Priority) = **Algorithm 3.3** (SumOutliers, chg_{CPU}, chg_{Th}, y_i^{CPU}, y_i^{Th})
10: ***for*** each mapped change type
11: Compute the number of repetitions of equal and successive changes:
(Repeat, start) = **ComputeOccurrence** (Type, vartype)
12: ***end for***
13: Historic data*:* Hy^{CPU}=[Hy^{CPU}; y_i^{CPU}], Hy^{Th}=[Hy^{Th}; y_i^{Th}]
14: Determine cycles and large deviations for both metrics
(LCycles, LLargeDeviationDown, LLargeDeviationUp)= **Algorithm3.4** ***(***Hy^{CPU}***)***
15: *(HCycles, HLargeDeviationDown, HLargeDeviationUp)***= Algorithm 3.4** ***(***Hy^{Th}***)***
16: Determine the type and the priority of notifications to send to resource planning: (Notification, Priority)= **Algorithm 3.5** (y_i^{CPU}, Type, Priority, Repeat, Start*, LLargeDeviationDown, LLargeDeviationUp,, HLargeDeviationDown, HLargeDeviationUp*)
17: return Notification, Priority
18: ***end for***

Table 3.2 Terms, definitions and symbols/acronyms

Symbol /Acronym	**Definition**
y	Observed data (e. g. , y^{CPU} , y^{Th})
$\hat{y}$	Predicted data (e. g. , $\widehat{y^{CPU}}$, $\widehat{y^{Th}}$)
chg^{CPU} , chg^{Th}	Type and position of change of detected outliers (CPU, throughput, respectively) in each time interval
out^{CPU} , out^{Th}	Detected outliers (CPU, throughput, respectively) in each time interval
$outlier^{CPU}$, $outlier^{Th}$	Detected outliers (CPU, throughput, respectively) in dataset
$RThs_{max}$	Maximum resource capacity threshold
$RThs_{min}$	Minimum resource capacity threshold
chgType	Type of change after metric mapping
Notification	Notification of proposed decision regarding resource and service metrics mapping
Priority	Priority of proposed decision
NRepetition	Number of repetitions of successive and similar types of change
m	Size of sliding window
Diver-Threshold	Threshold of Kullback-Leibler divergence
MAD	Median Absolute Deviation (*MAD*): *MAD*= median of $\{\| \boldsymbol{y_i} - \tilde{\boldsymbol{y}} \|\}$
$\tilde{y}$	Median of a giving dataset $(y_1, y_2, \dots, y_n)$
C_{ij}	Number of elements that actually belong to class C_i and are classified as class C_j (element of Confusion Matrix)
l	Number of classes (types of change)
tp_i	Number of correctly recognized class (true positives): C_{ij}
fp_i	Number of incorrectly assigned to the class (false positives): $fp_i = \sum_{j=1}^{l} C_{ji} - tp_i$
fn_i	Number of cases incorrectly unrecognized as belonging to the class (false negatives): $fn_i = \sum_{j=1}^{l} C_{ij} - tp_i$
tn_i	Number of correctly recognized cases that do not belong to the class (true negatives): $tn_i = \sum_{j=1}^{l} \sum_{k=1}^{l} C_{jk} - tp_i - fp_i - fn_i$
$Accuracy_i$	$(tp_i + tn_i)/ (tp_i + fp_i + fn_i + tn_i)$
$Precision_i$	$tp_i / (tp_i + fp_i)$
$Sensitivity_i$	$tp_i / (tp_i + fn_i)$
CPS	Call Per Second
MPS	Message Per Second

3.5 Detection of unusual changes

3.5.1 Definitions

According to Hawkins (Hawkins, 1980), “An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.” Outliers are individual data points that are inconsistent with the remaining data.

The recognition of outliers in data generated by a system is crucial in distinguishing its normal from abnormal behavior and helps to understand the causes of such an unusual behavior and to address it. The detection and the analysis of outliers are used in various applications; for instance, (Aggarwal, 2015) covers intrusion detection in systems, credit card fraud, interesting sensor events, and medical diagnosis. Most outlier detection methods create a model of normal patterns or behaviors. Anomalies are recognized as deviations from this normal model. The degree of deviation of a data point is quantified by a numeric value, known as the outlier score (Aggarwal, 2015). Accordingly, most outlier detection algorithms provide a score that can be one of two types (Aggarwal, 2015): real-value outlier score, which quantifies the tendency for a data point to be an outlier (compared to a given threshold or a given data point); or binary label, which indicates whether or not a piece of data is an outlier. Depending on various such as the type of the systems, the availability and the type of data, outlier detection can be performed using several models; for instance, (Aggarwal, 2015) covers clustering models, distance-based models, density-based models, probabilistic models and information-theoretic models.

In this section, we present the unusual changes detection techniques used in the proposed algorithm namely, Kullback-Leibler divergence, which features information-theoretic models, and Modified Z-Score, which is a distance-based model.

3.5.2 Kullback-Leibler divergence

To measure the divergence between the observed service and resource level metrics values (e.g., throughput and CPU usage) in each data window, we use the Kullback-Leibler divergence, also called relative entropy. It is a measure of the distance between two probability distributions on a random variable. Given two probability distributions *P(y)* and *Q(y)* over a discrete random variable y, the Kullback-Leibler divergence ($D_{KL}(P||Q)$) can be defined as follows (Shyam & Manvi, 2016) (Braverman, 2011) (Jing et al., 2013):

$$D_{KL}(P\,||Q) = \sum_{y \in Y} P(y) log_2 \frac{P(y)}{Q(y)} \tag{3.1}$$

In each sliding window, we divide the collected data (e.g., CPU usage and throughput) into two datasets; the first one is considered as the distribution *Q(y)* and the second one as *P(y)*. Next, we compute $D_{KL}(P\,||Q)$. The smaller the value of D_{KL}, the more similar *P(y)* is to *Q(y)* (stability in workload). Yet, if D_{KL} is greater than a given threshold (e.g., 0.1), we consider that *P(y)* is not similar (divergent) to *Q(y)*, and we designate this potential divergence as a possible presence of outliers in data which denotes significant changes in the system state.

3.5.3 Modified Z-Score

When a divergence is detected using Kullback-Leibler, both the type of the significant change, whether it is an increase or decrease in the workload, and the time at which this spike occurred (i.e., position in data window), should be identified. To detect such workload variations and identify their types, we use the Modified Z-Score technique, which is an outlier labeling method. An observation is labeled as an outlier when its modified Z-Score exceeds an absolute value of 3.5. The Modified Z-Score is defined as follows (Iglewicz & Hoaglin) :

$$M_i = \frac{0.6745\,(y_i - \tilde{y})}{MAD} \tag{3.2}$$

where y_i is observed data, $\tilde{y}$ is the median of the dataset, and MAD is Median Absolute Deviation.

Algorithm 3.2 KL-Divergence

KL-Divergence (y_i^{CPU}, y_i^{Th}, *m, DiverThreshold*)

Input: (y_i^{CPU}, y_i^{Th}, *m, DiverThreshold*)
Output: div_{CPU}, div_{Th}, chg_{CPU}, chg_{Th}

1: Initialize arrays of change types of detected outliers: chg_{CPU}:=zero[], chg_{Th}:= zero[]
2: Define distributions: P_{CPU} and Q_{CPU}
3: Define distributions: P_{Th} and Q_{Th}
4: Compute Kullback-Leibler divergence: div_{CPU} = **KLDiv**(P_{CPU}, Q_{CPU})
5: Compute Kullback-Leibler divergence: div_{Th} = **KLDiv**(P_{Th}, Q_{Th})
6: ***if*** (div_{CPU}> *DiverThreshold*) ***or*** (div_{Th}> *DiverThreshold*)
7: Compute Modified Z-Score: $mzscore_{CPU}$ = **M-ZScore**(y_i^{CPU})
8: Compute Modified Z-Score: $mzscore_{Th}$ = **M-ZScore**(y_i^{Th})
9: Determine the change type: chg_{CPU}=**Variation**(div_{CPU}, $mzscore_{CPU}$, y_i^{CPU})
10: Determine the change type: chg_{Th} =**Variation**(div_{Th}, $mzscore_{Th}$, y_i^{Th})
11: ***end if***
12: return div_{CPU}, div_{Th}, chg_{CPU}, chg_{Th}

Algorithm 3.2 describes the proposed strategy for detecting abnormal behaviors, considered as outliers, in the system using the Kullback-Leibler and Modified Z-Score techniques. The algorithm starts by initializing the arrays of abnormal change types for the detected outliers (Line 1) and defining the distributions P_{CPU} and Q_{CPU} for resource metric data (e.g., CPU usage) and P_{Th} and Q_{Th} for service metric data (e.g., throughput) (Lines 2 and 3). Then, the Kullback-Leibler divergence is computed for both metrics (Lines 4 and 5). If the Kullback-Leibler divergence (resource metric or service metric) is greater than a given divergence threshold (e.g., threshold = 0.1), the modified Z-Scores are calculated and assigned to its corresponding metrics data position (Lines 7 and 8). By comparing the modified Z-Scores and the metrics data, Algorithm 3.2 determines the type of abnormal changes of detected divergences (outliers) and returns the results in the arrays chg_{CPU} and chg_{Th} (Lines 9 and 10). For instance, when an abnormal increase in CPU usage is detected at position *i*, the value 1 is assigned to chg_{CPU} at the corresponding position. On the other hand, when an abnormal decrease in CPU usage is detected, the value -1 is assigned to chg_{CPU}. Finally, Algorithm 2 returns the computed Kullback-Leibler divergence and the detected abnormal changes for both

metrics, which will be used for mapping and analyzing system changes and to propose the appropriate decisions to be taken accordingly.

3.6 Mapping

To determine the type of detected changes occurring in both the resource level and service level metrics during a giving time slot, we verify the mapped changes of detected outliers. We also analyze and compare the metrics' variations to identify the variation types and the priorities accordingly. For instance, if the mapped change value is equal to zero, this implies two possibilities, namely, an opposite change case or a steady situation. Indeed, in opposite change cases, we have to validate which one of the metrics is increasing to specify the exact type of change and its priority. If the variation of the resource level metric is an increase (metric variation =1), we assign a high priority type 3 to the change type and a warning notification is generated. Otherwise, we assign a high priority type 4 and a warning notification is created. In Table 3.3, we detail all possible cases of mapped changes, their designed type, priority, and the proposed notifications for each situation. These notifications range from warnings to resource adaptation notifications.

Table 3.3 Mapping of resource level and service level metrics' variations and their corresponding types, priorities and notifications

Metric Variation		Mapping	Type of Change	Priority	Notification
Resource	Service				
1	1	2	1	1	Resource adaptation (allocation)
-1	-1	-2	2	1	Resource adaptation (release)
1	-1	0	3	1	Warning (opposite change)
-1	1	0	4	1	Warning (opposite change)
0	-1	-1	5	2	Workload decrease
-1	0	-1	6	1	Warning (workload decrease)
0	1	1	7	2	Workload increase
1	0	1	8	1	Warning (workload increase)

0: no change/no outlier is detected
1: increasing workload is detected
-1: decreasing workload is detected

3.7 Periodicity detection

The Periodicity detection algorithm, proposed by St-Onge et al. (St-Onge, Kara, et al., 2020), aimed to detect patterns of varying lengths, amplitude, and shapes for cloud computing systems. It is composed of two essential steps, namely, extraction of digital prints from collected data, and detection of patterns and cycles. In the first step, the data are processed using a binary conversion and a prefix transversal operation to obtain digital prints. In the second step, the digital print deviation for each data point is computed and compared to predefined pattern deviation shapes to detect periodic patterns.

In this paper, we propose an advanced version of the periodicity detection algorithm. It aims to strengthen the notification mechanism by providing detected cycles and large deviations in data, which are compared to our mapping algorithm results (based on outlier detection techniques) to make notification decisions. In the initial version of the periodicity detection algorithm, data are processed offline, and only detected patterns are provided, while our advanced version offers an online data processing over a sliding window, and is able to detect periodic patterns as well as unusual behaviors characterized by a large deviation of analyzed metrics. These large deviations are extracted using an adaptive weighting of predefined patterns. Moreover, we replace continuous spline generation with the moving average to get discretized mean values $\mu(t)$ because the latter do not require a lot of data samples, which is more appropriate for online processing.

In the following sections, we present the preparation phase algorithm and the pattern detection algorithm. We also give a brief description of the concepts of moving average, prefix transposition, digital prints, digital print deviation and evaluation patterns.

3.7.1 Preparation phase

3.7.1.1 Moving average

The moving average of a giving time series data is calculated over a sliding window of length *n* across neighboring elements. This helps reduce the effect of temporary variations in data and show the data's trend, as well as the values above or below the trend. The moving average is defined as a sequence given by $\{a_i\}_{i=1}^{N}$, where a moving average MA_n is a new sequence $\{S_i\}_{i=1}^{N-n+1}$ defined from a_i, taking the arithmetic mean of the sequence of *n* terms, such that (León-Castro, Avilés-Ochoa, & Merigó, 2018; WebFinance):

$$S_i = \frac{1}{n}\sum_{j=i}^{i+n-1} a_j \tag{3.3}$$

3.7.1.2 Prefix transposition

In the field of molecular biology, prefix transposition is commonly used to infer evolutionary and functional relationships in genomes. Its underlying mechanism is based on the fact that the transposition distance between two permutations can be used to estimate the number of global mutations between genomes. In this sub-section, we explain how to adapt this concept to a periodicity detection problem.

A transposition operation involves swapping two adjacent substrings in a string. Since, in this work, we apply prefix transpositions on data samples converted into binary strings, a prefix transposition operation $f_p(1, x, y)$ applied to a string $s = [s[n], s[n-1] \dots, s[2], s[1]]$ of length *n*, where $1 < x < y \leq (n+1)$, will perform a rearrangement event that transforms s into $[s[n], \dots, s[y], s[x-1], \dots, s[1], s[y-1], \dots, s[x]]$.

For example, say we want to apply the prefix transposition operation $f_p(1,3,5)$ on the string $s = 100101101$. Now, $s[n], \dots, s[y] = s[9], \dots, s[5] = 10010$, $s[x-1], \dots, s[1] =$

$s[2], \dots, s[1] = 01, \quad s[y-1], \dots, s[x] = s[4], \dots, s[3] = 11$. Therefore, we gets = 10010**0111**.

Before carrying out a prefix transposition operation $f_p(1, x, y)$, it is necessary to estimate the parameters *x* and *y*. In the proposed approach, the process of estimating these parameters is entirely automated and given by the following equations:

$$\sigma^2 = Var(|X_t - \mu(t)|) \tag{3.4}$$

$$x = \lfloor log_2(\sigma) \rfloor \tag{3.5}$$

$$\delta X_{max} = max(|X_{t+1} - X_t|) \tag{3.6}$$

$\forall\, t = 1,2, \dots, n-1$

where,

X_t: Observed data sample

n: Size of the dataset

$$y = x + \left\lfloor log_2 \left(\left\lceil \frac{\delta X_{max}}{2^{x-1}} \right\rceil \right) \right\rfloor + 2 \tag{3.7}$$

It is relevant to mention that the equations of *x* and *y* are modified when the data sample is stable or uniform to avoid null or negative values. For instance, the equation of *x* becomes:

$$x = \lfloor log_2(\sigma) \rfloor + 2 \tag{3.5'}$$

3.7.1.3 Digital print

The digital printing operation is an approach proposed as an adjustment to the transposition process. Its function is to help determine the maximum number of step deviations, as well as the grid size of the workload's transpositions, and is a key component of the proposed approach. The digital print is a sub-string defined by $\rho_t = [s[y-1], \dots, s[x]]$, extracted following a prefix transposition operation $f_p(1, x, y)$. The grid size and maximum number of step deviations of a digital print dataset are obtained using the following equations:

Grid size:

$$2^{y-x} \tag{3.8}$$

Max step:

$$2^{y-x-1} - 1 \tag{3.9}$$

3.7.1.4 Preparation phase algorithm

Algorithm 3.4.1 presents the proposed process for preparing collected data that will be evaluated by the pattern detection algorithm. It starts by generating the moving average (Line1) for input data and returns a discretized $\mu(t)$. If observed data are the CPU rate, the $\mu(t)$ values are multiplied by 100 to get discretized values (named *cpuRate*) ranging from 0 to 10,000 (Line 2). In the next step, the *cpuRate* array is converted to binary strings (Line 3), and used along with observed data to compute *x* and *y* values for prefix transreversal transformation (Line 4 to 6). Afterwards, the binary strings, *x* and *y* parameters are evaluated to select positive and negative evaluation patterns (Line7) and to get transposed binary strings and digital prints using prefix transreversal operations (Line 8). The normalized strings and reduction length are then obtained by the applying l-transreversal process (Line 9). Next, the algorithm computes the grouping distance of the normalized strings to get binary pairs and the number of 2-transreversal and l-transreversal operations (Line 10). Finally, the algorithm provides *x* and *y* parameters, the evaluation patterns and the digital prints (Line 11) which will be used by the advanced detection periodicity algorithm, described in the next section.

The preparation phase is a method whose time complexity mainly depends on two parameters: the size of the data collection history *h*, which increases at each iteration of the preparation phase method up to a fixed size chosen by the user, and the length *l* of the binary string being evaluated. First, the collected data are appended to a sliding window to evaluate a moving average value, thus having a time complexity *O(h)* (Line 1). In the case of CPU rate evaluation, we multiply the data by 100, with a time complexity *O(1)* (Line 2). The data are then converted into a binary string of size *l* and appended to a binary string history *binStr[]*, with a time complexity *O(h.l)* (Line 3). Next, a series of computations on the moving average history are

Algorithm 3.4.1 Preparation Phase

Preparation Phase (*data[]*)

Input: *data[]*
Output: *x, y, evalPatterns[], digitalPrint[]*

1: Generate moving average mu:
   mu = moveAverage(data[])
2: If CPU rate, multiply by 100:
   *cpuRate= mu*100*
3: Convert to binary strings:
   binStr[] = decTobin (*cpuRate[]*)
4: Get *x, y* and evaluation patterns:
   x=floor(log2(std(abs(100*(data[]-mu[])))))
5: *maxDelta = max*(abs(*mu[i+1]- mu[i])*)
6: *y = x+ floor(log2(ceil((100*maxDelta)/(2^(x-1)))))+2*
7: *evalPatterns[] = getEvalPatterns(x,y,binStr[])*
8: Get transposed binary strings and digital prints*:*
   (transposedStr[],digitalPrint[])=prefixTransreversal(x,y,binStr[])
9: Get normalized strings and reduction length by applying l-Transreversal:
   (normStr[],l[]) = lTransreversal(transposedStr[])
10: Computing the grouping distance to get a binary pair, 2-trans and l-trans distances:
   (binPair[],twoTrans[],oneTrans[])=groupingDist(normStr[])
11: **return** *x, y, evalPatterns[], digitalPrint[]*

performed (Lines 4,5,6) to get the *x* and *y* values necessary for prefix transreversal, each respectively with a time complexity of *O(h), O(h), O(1)*. The algorithm proceeds with a method *getEvaluationPatterns()* returning an array of evaluation patterns (Line 7), at time complexity *O(h·l)*. The last actions are then to successively apply prefix transreversal, l-transreversal and grouping distance operations (Lines 8, 9, 10) on the data, then appending the results to the corresponding arrays, each with a time complexity *O(h·l)*. Hence, the time complexity of the preparation phase algorithm is *O(h) + O(1) + O(h.l) + O(h) + O(h) + O(1) + O(h·l) + O(h·l) + O(h·l) + O(h·l)*, which is equivalent to *O(h.l)*.

3.7.2 Advanced periodicity detection phase

3.7.2.1 Digital print deviation

Digital print deviations offer useful insights into a dataset's trends and patterns. It is by analyzing these deviations that the proposed approach detects variations in shape, amplitude and length in a dataset. The digital printing process is achieved by subtracting each digital print value at time *t + 1* by its predecessor at time *t*, as shown below:

$$\delta\rho_t = \rho_{t+1} - \rho_t \quad (3.10)$$

$\forall\, t = 1,2, \ldots, n-1$

3.7.2.2 Evaluation patterns

Evaluation patterns are binary strings defining deviation shapes (or trends) that are compared, through sliding windows, with the digital print deviations of a dataset. Different evaluation pattern sets can be defined by a user, suiting any preference of shape, amplitude or length that the proposed approach would be required to detect in a dataset. As such, evaluation patterns can be defined following the following parameters:

1. Slope sharpness (e.g., sharp slope: smaller string size, sharper deviations, smooth slope: longer string size, smoother deviations),
2. Positive or negative deviation,
3. Proximity of all digital print deviation values to those of the evaluation pattern in a sliding window (e.g., all digital print deviation values must be equal to or greater than those of an evaluation pattern).

For example, the proposed approach could be required to find consecutive pairs of positive and negative deviation patterns, thus detecting a cycle in the process. One way to do so would be to define two evaluation patterns: an evaluation pattern to detect smooth positive deviations $s_1 = [0,0,1,1]$ and another evaluation pattern to detect sharp negative deviations $s_2 = [-1,-3]$. We would then let the periodicity detection algorithm screen the dataset with two sliding windows such that:

$$[\delta\rho_t, \delta\rho_{t+1}, \delta\rho_{t+2}, \delta\rho_{t+3}] \geq s_1 \quad (3.11)$$

$\forall\, t = 1,2, \ldots, n-3$
and

$$[\delta\rho_t, \delta\rho_{t+1}] \leq s_2 \quad (3.12)$$

$\forall\, t = 1,2, \ldots, n-1$

Each consecutive pair of similar patterns $[s_1, s_2]$ would therefore form a cycle that can later be analyzed to obtain its length, amplitude and shape.

3.7.2.3 Weighted evaluation patterns

To detect large variations in collected data using the periodicity detection algorithm, we propose to detect and evaluate the large deviation using evaluation patterns weighting. The weight is computed dynamically at each evaluation step, based on the standard deviation of computed deviation values. It is defined as follows:

$$weight = round(std(deviation[])) + 1 \quad (3.13)$$

The computed weight is then multiplied by the positive evaluation pattern and negative evaluation pattern for detecting the large positive deviation and large negative deviation, respectively, such as, for example:

$$[\delta\rho_t, \delta\rho_{t+1}, \delta\rho_{t+2}, \delta\rho_{t+3}] \geq weight * s_1 \quad (3.14)$$

$\forall\, t = 1,2, \ldots, n-3$
and

$$[\delta\rho_t, \delta\rho_{t+1}] \leq weight * s_2 \quad (3.15)$$

$\forall\, t = 1,2, \ldots, n-1$
where,
s_1: Evaluation pattern for positive deviations
s_2: Evaluation pattern for negative deviations

3.7.2.4 Advanced periodicity detection algorithm

Algorithm 3.4 shows the proposed periodicity detection process. The algorithm starts by calling the preparation phase algorithm to get the *x* and *y* parameters, the evaluation patterns and the digital prints (Line 1). The grid size and the maximum step are then initialized (Line 2 and 3), and the binary values of the digital prints are converted to decimals (Line 4). Next, each consecutive digital print value is evaluated to generate a deviation matrix (Line 5). After that, the *evalPatterns[]* array is divided into two separate attributes: the first one for the positive evaluation pattern and the second one for the negative evaluation pattern (Line 6). The next step computes a weight value that will be multiplied by the evaluation pattern windows for detecting large deviations (Line 7). Afterwards, the algorithm evaluates the digital print deviations using two distinct evaluation windows. The first evaluation window assesses the digital print deviation with positive and negative patterns (*windowUp*, *windowDown*) to detect patterns (Line 8), while weighted positive and negative patterns are used to detect large deviations (Line 9). Next, pairs of positive and negative deviations are evaluated to determine their locations in the time series (Line 10), and matching pairs are identified as sequential cycles (Line 11). Finally, the algorithm returns detected cycles and detected large positive and large negative deviations (Line 12).

Time complexity of the Advanced Periodicity Detection algorithm depends on the array size of the collected data history, as well as the string length of attributes such as digital print values. As such, the first step is to append a new collected data to the data history and pass that data history array to the preparation phase algorithm (Line 1), with a time complexity of $O(h.l)$, where l is the binary string length. Then, *gridSize* and *maxStep* values are evaluated, given the newly obtained *x* and *y* values (Lines 2, 3), each with a time complexity of $O(1)$. The digital print values are then converted to decimal values (Line 4) with a time complexity of $O(h.l)$. Next, the digital prints and deviations are processed (Lines 5, 8, 9) to detect trends and behaviors of the collected data, with each loop having a time complexity of $O(h)$. Throughout the Advanced Periodicity Detection algorithm, there are also instances where evaluation

Algorithm 3.4 Advanced Periodicity Detection

Advanced Periodicity Detection (*data[], proximity)*

Input: (*data[], proximity*)
Output: *cycle[]*, *LargeDeviationUp[], LargeDeviationDown*[]

1. *(x, y, evalPatterns[], digitalPrint[])*= **Preparation Phase** (*data[]*)
2. Initialize grid size and max step number:
   *gridSize = 2^*(*y-x*)
3. *maxStep = (gridSize/2)-1*
4. Convert digital prints to decimal:
   decPrint = binTodec(digitalPrint[])
5. Build deviation matrix : *deviation[]*
6. Prepare sliding window:
   windowUp = even(evalPatterns[])
   windowDown =odd(evalPatterns[])
7. Prepare weight for Large deviation detection using sliding window weighting:
   weight:= round(std(deviation[]))+1
8. Detect consecutive pairs of positive and negative deviation patterns using evaluation patterns (*windowUp, windowDown*): *detectedDeviationUp[], detectedDeviationDown[]*
9. Detect large positive and/or large negative deviation patterns using evaluation patterns weighting : *LargeDeviationUp[], LargeDeviationDown[]*
10. Evaluate pairs of positive and negative deviations:
    (*posC[], lengthC[], shapeC[]*)= evalDeviations (*detectedDeviationUp[], detectedDeviationDown[]*)
11. *cycle*=[*posC[], lengthC[], shapeC[]*]
12. ***return*** *cycle[]*, *LargeDeviationUp[], LargeDeviationDown*[]

patterns are split into positive and negative evaluation windows (Line 6) and a weight is evaluated for large deviation detection (Line 7). These evaluations have a time complexity of *O(h)*. Further, the evaluation of detected deviations (Line 10) to determine cycles (length, shape and amplitude) is performed by using two nested loops. The first spans from i=1 to h, and the second from $j=i+1$ to h, thus having a time complexity of $O(h \cdot log(h))$. Finally, the detected cycles are identified (Line 11) and returned as well as the detected large deviations (Line 12), with a time complexity *O(1)*. Hence, the time complexity of the whole algorithm is $O(h.l) + O(1) + O(h.l) + O(h) + O(h) + O(h) + O(h) + O(h) + O(h \cdot log(h)) + O(1)$, which is equivalent to *O(h.l)*.

3.8 Notification

In this section, we discuss the notification mechanism of our abnormal behavior detection solution. This process realizes the analysis of outlier detection and periodicity detection results, the monitoring of resource utilization compared to the maximum capacity threshold and the minimum capacity threshold of available resources, and if necessary the generation of notifications to trigger the resource adaptation or to alert the monitoring and control system about unusual system behavior. These notifications are sent to the resource planning component which evaluates the current state of the system, the predicted resource consumption, the received notifications regarding unusual changes, the repetition and the priority of detected changes, and the detected cycles. The resource planning then estimates the amount of required resources and the available time to deploy them.

3.8.1 Resource capacity thresholds

To avoid wastage due to resource idling or system failure due to lack of resources, we define two resource capacity thresholds, namely, *minCapacity* and *maxCapacity.* Therefore, we consider 25% of available resources as a threshold for minimum capacity, and 80% of available resources as a maximum capacity threshold. These thresholds' values were established based on our industrial partners' propositions. For example, let us assume that the available resources are 3 CPU cores (300%); a notification for the resource adaptation is then generated whenever the CPU utilization is smaller than the *minCapacity* threshold (75%) or is greater than the *maxCapacity* threshold (240%).

The definitions of *minCapacity* and *maxCapacity* are given by the following equations:

$$minCapacity = 0.25 * availableResources \quad (3.16)$$

$$maxCapacity = 0.8 * availableResources \quad (3.17)$$

3.8.2 Tolerance ratios

During the notification process, our proposed algorithm computes the ratios between the resource utilization and the capacity thresholds (the minimum and the maximum capacity thresholds, respectively), and compares them to giving thresholds. It aims to check if the system consumption approaches the idle state or the overload state and generates high priority notifications if they occur. It is interesting to note that the greater the given threshold, the lower the tolerance for the resource consumption to get to the minimum or the maximum capacity. For our experimental evaluation, we set this threshold to 2.

Therefore, we define two ratios, namely, *minRatio* and *maxRatio* as follows:

$$minRatio = round(y_i^{CPU} / minCapacity) \quad (3.18)$$

$$maxRatio = round(maxCapacity / y_i^{CPU}) \quad (3.19)$$

3.8.3 Notification algorithm

Algorithm 3.5 shows the proposed notification generation process. It starts by initializing the filtered notifications and the system capacity overtaking arrays (Line 1). The next step examines the mapped variation types and compares current resource consumption to the maximum and the minimum resource capacities (Line 2). Therefore, each change type detected by our mapping algorithm, its priority, its category (warning, increase type or decrease type), and its position in collected data are identified. If the variation type has a high priority (e.g., warning types; decreasing type 2, and increasing type 1), or the maximum/minimum resource capacity threshold is reached, a high priority notification is filtered (Line 3). Otherwise, the algorithm examines the deviation, the detected variation occurrences and the resource consumption tolerance ratio to whether or not to raise the variation type priority (Line 4). For instance, if the variation type is an increasing type, with a low priority (change type 7), the algorithm verifies the availability of resources, the significance of this variation, its repetition and trend. If the tolerance ratio is less than the specified threshold and at least one metric deviation is significant (large deviation detected) or it is a repeated variation, the notification

is filtered with high priority, reflecting a strong increasing tendency. Finally, the filtered notifications, their positions and their priority are returned (Line 5).

Algorithm 3.5 Notification

Notification (y_i^{CPU}, *Type, Priority, Repeat, maxCapacity, minCapacity, warningType, decreaseType, increaseType LLargeDeviationDown, LLargeDeviationUp, HLargeDeviationDown, HLargeDeviationUp, threshold, NType*)

Input: (y_i^{CPU}, *Type, Priority, Repeat, maxCapacity, minCapacity, warningType, decreaseType, increaseType LLargeDeviationDown, LLargeDeviationUp, HLargeDeviationDown, HLargeDeviationUp, threshold, NType*)
Output: *FilterNotif*, *Priority*, *OverCapacity*

1: Initialize arrays of filtered notifications and overtaking of system capacity: *FilterNotif* =zero[], *OverCapacity*= zero[]
2: Examine mapped variation types, their priorities and compare resource consumption to max/min resource capacity thresholds
3: Filter high priority notifications (warnings, decreasing type, increasing type) and system capacity overtaking cases
4: Examine deviation, detected variation occurrences and resource consumption tolerance ratio to whether or not to raise variation type priority, and filter corresponding notifications
5: ***return*** *FilterNotif*, *Priority*, *OverCapacity*

The time complexity of the notification algorithm depends on the number of mapped variation types. First, the initialization of arrays (notification and capacity threshold overtaking arrays) has a time complexity *O(1)* (Line 1). Next, the search of each defined change type has a time complexity *O(n),* where *n* is the size of the collected data (Line 2). The filtering of notifications based on the change type, its priority and the resource consumption have a time complexity $O(Tn^2)$, where *n* is the size of the collected data, *T* is the change type number (Lines 2 to 4) and $T<n$. Indeed, the verification of the type and the priority has a time complexity *O(s),* where *s* is the size of the change subtype (e.g., warning types, increasing types) and $s<T$, and the deviation verification (Line 4) has a time complexity *O(n).* The results assignment of notifications, priorities and capacity overtaking to corresponding arrays is performed with a

time complexity *O(1)* . Hence, the time complexity of the notification algorithm is *O(1)* + *T* [*O(n)*+ *n[O(s)*+ *O(1)*+ *O(n)*+ *O(n)]]*+ *O(1)* which is equivalent to $O(Tn^2)$.

3.9 Experimental evaluation

In this section, we present the results of the experiments using our proposed algorithm. By processing data from various systems and load profiles, we aim to evaluate the ability of our algorithm to accurately detect abnormal variations and provide notification regarding the types of these variations. Therefore, we use data from the OpenIMS telecommunication service platform (Configurations 1-5 presented in Table 3.5), which represent CPU usage and throughput collected while testing several configurations, and data furnished by the Ericsson Company (Datasets 1, 2, 3 and 4), which constitute CPU usage, memory usage, throughput and latency collected from test runs with different CPU configurations of a telecom system. Next, we define the evaluation metrics we used to measure the accuracy of our algorithm, we present the experimental setting and data, and we analyze the obtained results.

3.9.1 Evaluation metrics

The accuracy evaluation of our proposed algorithm (Algorithm 3.1) aims to measure its ability to correctly detect each abnormal behavior and recognize its type. Therefore, we apply the evaluation techniques used for multi-class classification (Dunn, 2014) (Sokolova & Lapalme, 2009). First, we assume that each detected abnormal behavior belongs to one of *n* different classes, and we consider every change type T_i (type: 1, 2, 3, …, 8) as a class C_i (class label: 1, 2, 3, …, 8). Then we compute the number of correctly recognized change type samples (true positives: *tp*), the number of correctly recognized change type samples that do not belong to the evaluated change type (true negatives: *tn*), the number of incorrectly assigned change type samples (false positives: *fp*), and the number of incorrectly unrecognized change type samples as belonging to the evaluated change type (false negatives: *fn*) (Sokolova & Lapalme, 2009). These four numbers are computed for each change type (class) using the Confusion Matrix (Intel Developer) shown in Table 3.4. The elements C_{ij} of the Confusion Matrix denote the

number of elements that actually belong to class C_i, and are classified as class C_j. For instance, C_{22} represent the number of elements that actually belong to change type 2 (T_i = 2) and are marked by our algorithm as change type 2. The assessment for an individual class C_i (change type T_i) is defined by tp_i, fn_i, tn_i, fp_i, $accuracy_i$, $precision_i$ and $sensitivity_i$. Table 3.2 details the formulas used for each of the proposed evaluation metrics.

Table 3.4 Confusion Matrix for multi-class classification

	Classified as Class C_1	…	**Classified as Class C_i**	…	**Classified as Class C_l**
Actual Class C_1	C_{11}	…	C_{1i}	…	C_{1l}
…	…	…	…	…	…
Actual Class C_i	C_{i1}	…	C_{ii}	…	C_{il}
…	…	…	…	…	…
Actual Class C_l	C_{l1}	…	C_{li}	…	C_{ll}

3.9.2 Experimental Setting

3.9.2.1 Testbed setup

The testbed was built with 4 servers, all connected to the same local area network. For IMS Core components, we used a 3.6 GHz CPU, 24 GB of RAM, and the Linux Ubuntu operating system, version 14.04. The testbed OpenIMS Core (Fokus, 2014) was an open-source implementation of IMS Core developed by Fraunhofer FOKUS IMS Bench. The virtualization of IMS Core (CSCFs and HSS) is based on the Linux container technology (Containers, 2017). Further, a SIP client was created using the instantiation of SIPp, version 591 (October 2010), which is a traffic generator (Gayraud & Jacques, 2014).

3.9.2.2 Data / Load Profile

The results of a performance analysis conducted on OpenIMS showed that S-CSCF forms a bottleneck (Mkwawa & Kouvatsos, 2008). Therefore, we use S-CSCF in the following evaluation process. On the one hand, we use the CPU consumption as a resource level metric

for this analysis, since it has a significant effect on the performance (Liang et al., 2011). Further, we use the throughput of a virtualized S-CSCF node as a service-level metric. It reflects the number of ringing messages received by that node.

We perform several tests on our OpenIMS platform using different load profiles. Particularly, Configuration 2 includes a sharp decrease in workload, while Configuration 3 involves a sharp decrease, followed by a sharp increase in the workload. Varying the type (increase/decrease) and the significance (sharp/smooth) of changes in the workload profiles, we aim to assess our algorithm to accurately detect abnormal behaviors. We also aim to test our algorithm in case of periodic system behavior and significance variations. Therefore, we define multiple variants of selected configurations (e.g., Configuration2) and we test them on our OpenIMS platform. The collected data are then concatenated to provide periodic and varying workload data, namely, Configuration 4 and Configuration 5. In these experiments, the CPU usage and the throughput were collected every 5 seconds. The descriptions of Configuration 1, Configuration 2, Configuration 3, Configuration 4 and Configuration 5 are presented in Table 3.5, while the configuration variants are described in Table 3.6 and Table 3.7.

Table 3.5 Description of load profiles (OpenIMS platform)

Load Profile	Description
Configuration1 (Config1)	Start at 150 CPS, increments: 50 CPS/10 sec until 1200 CPS, decrement: 50 CPS/10 sec until 150 CPS.
Configuration2 (Config2)	Start at 150 CPS, increment: 50 CPS/10 sec until 400 CPS, 400 CPS constant during 100 sec, 600 CPS constant during 300 sec, 200 CPS decrement: 50 CPS/50 sec until 50 CPS.
Configuration3 (Config3)	Start at 150 CPS, increment: 50 CPS/10 sec until 400 CPS, 400 CPS constant during 100 sec, 600 CPS constant during 300 sec, 5 CPS during 60 sec, 400 CPS constant during 300 sec.
Configuration4- (Config4)	The 5 cycles are variations of Configuration 2.
Configuration5 (Config5)	The 8 cycles are variation of the following configuration: start at 150 CPS, increment: 50 CPS/50 sec until 300 CPS, 500 CPS increment: 50 CPS/50 sec until 600 CPS, 900 CPS increment: 50 CPS/50 sec until 1050 CPS.

Table 3.6 Config4, Configuration variations

Cycle	Variation
1	+ 50 CPS
2	-50 CPS
3	-100 CPS
4	-25 CPS
5	*300 CPS*

Table 3.7 Config5, Configuration variations

Cycle	Variation
1	-100 CPS
2	+275 CPS
3	-25 CPS
4	+400 CPS
5	+25 CPS
6	+100
7	+350
8	-50

The data provided by the Ericsson Company represent CPU consumption Dataset1 (Dset1), memory usage Dataset2 (Dset2), throughput Dataset3 (Dset3) and latency Dataset4 (Dset4) collected every 5 seconds during 24 hours from the Telecom system test runs with different CPU configurations. In this article, we present two examples of service-level-to-resource-level metrics mapping, i.e., CPU usage-to-throughput mapping and memory usage-to-latency mapping.

3.9.2.3 Assumptions

1. The monitoring tools are sufficiently accurate, and the collecting data from the system is continuous and in real-time.
2. Data for both service- and resource-level metrics is collected simultaneously and reflects the system state at both levels.
3. The Linux container-based virtualization and the CPU core isolation technique guarantee that collected data reflect the effective service/application consumption.

3.9.3 Results and analysis

In this section, we discuss the evaluation of the results of the experiments examining the ability of our algorithm to accurately detect and provide notifications regarding unusual system changes.

For each load profile defined (see Table 3.5), we tested the proposed algorithm and captured the metrics mapping values, types and position for unusual changes detected in every sliding window, as well as the cycles and large deviations, where required. Figure 3.2 presents the abnormal behavior detection results in Configuration 1. The workload profile in this configuration increases and then decreases progressively (50 CPS/10 sec), and it does not include any sudden and significant changes (see Figure 3.2.b). Hence, the mapping process does not detect abnormal behavior throughout the test (mapping = 0) (see Figure 3.2.a).

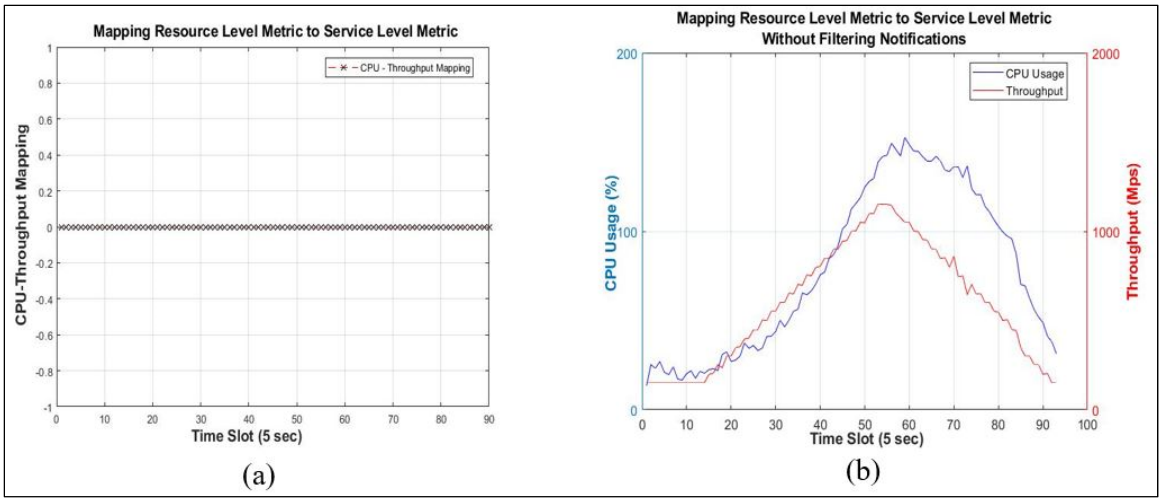

Figure 3.2 Config1 – Mapping and variation detection

When the workload gradually increases or decreases, our algorithm processes collected data, finds null values for the metrics mapping, and provides information that there are no significant system changes. Such situations are represented throughout all data in Configuration 1, as well as during the ascending phase (i.e., from the beginning of the test to the 100th collected data value) in Configurations 2 and 3 (see Figures. 3.4.a and 3.6.a). These results prove the ability of our algorithm to recognize and distinguish between normal and abnormal changes.

Next, we have to assess whether abnormal changes detected by our algorithm and their assigned types correspond to real changes in the workload profiles. Configuration 2 includes a progressive increase (50 CPS/10 sec) followed by a sudden and significant decrease (from 600 CPS to 200 CPS) in the workload profile, which reflects an unusual behavior as compared to the previous one. By processing collected data from the Configuration 2 experiment, our algorithm maps the throughput to the CPU usage and successfully detects two consecutive cases of decreasing workload, corresponding to -1 value (Figure 3.3) at positions 101 and 102, and interprets them as “decreasing workload” change type (type=5) (see Figure 3.4.a). Because of the successive repetition of the change type 5 at positions 101 and 102, a decreasing workload notification is generated and filtered (see Figure 3.4.b) to be sent to resource planning.

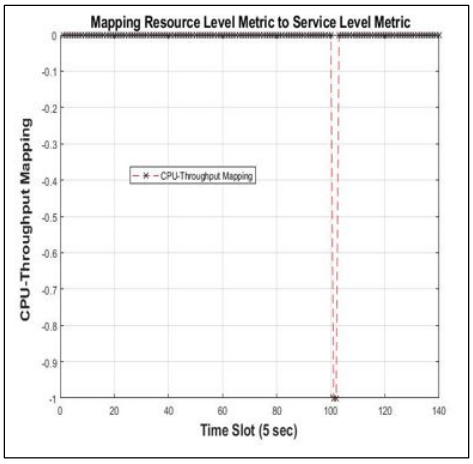

Figure 3.3 Config2 – Mapping

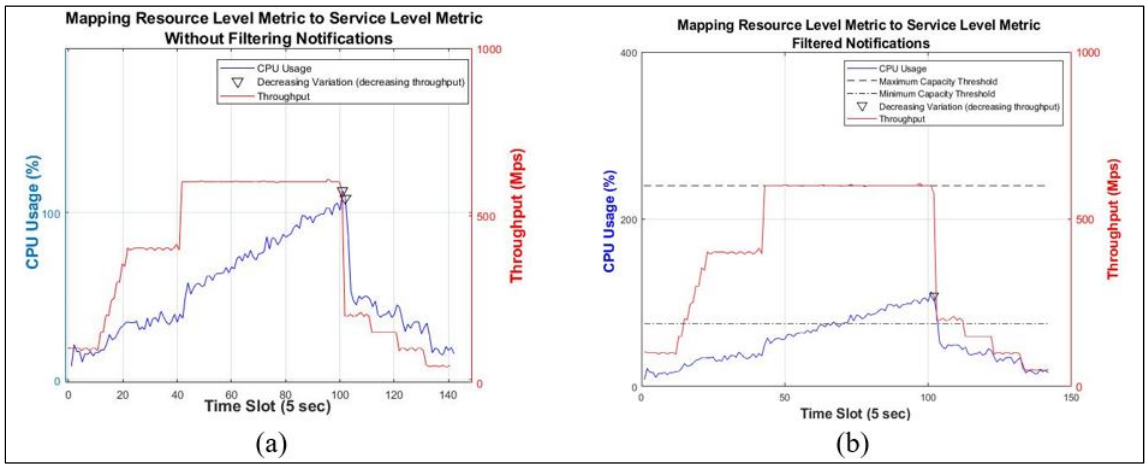

Figure 3.4 Config2 – Variation detection and filtered notifications

Furthermore, we evaluate our algorithm with successive and inconsistent changes in workload. Configuration 3 includes: a progressive increase (50 CPS/10 sec), a constant workload, a sharp decrease (from 600 CPS to 5 CPS), and a sharp increase (from 5 CPS to 400 CPS) in workload. The results show that in the sharp decrease phase, our algorithm was able to find three cases of unusual changes, and their corresponding mappings were 0, -2, -1 at positions 101,102, 103, respectively (see Figure 3.5). In the sharp increase phase, the algorithm found 0, 2, 1 as CPU usage-to-throughput mapping results at positions 111, 112, 113, respectively (see Figure 3.5).

Further, the change types assigned by our algorithm to the mapping results were 4, 2, 6 at positions 101,102, 103, respectively, and 4, 1, 8 at positions 111,112, 113, respectively (see Figure 3.6.a). Change type 4 dictates opposite changes in both metrics, which requires a warning notification. Such a behavior may be due to a certain delay in the time to react to workload changes between the service and the resource levels.

For instance, at the beginning of the decrease phase, the CPU usage decreases from 109% to 107%, while the throughput increases from 552 MPS to 600 MPS. The next collected data are mapped, and similarly involve notifications for resources adaptation. For instance, detected change type 2 at position 102 necessitates the release of resources, while detected change type 1 at position 112 requires resource allocation (see Figure 3.6.a). At the end of a sharp decrease phase (position 103), our algorithm detects a decrease in CPU usage, with a stable throughput (change type 6) and generates a warning error notification accordingly. Similarly, at the end of the sharp increase phase (position 113), our algorithm successfully detects an increase in CPU usage, with a stable throughput (change type 8) and provides a warning error notification. All detected change types are filtered to generate notifications since they have high priorities (see Figure 3.6.b).

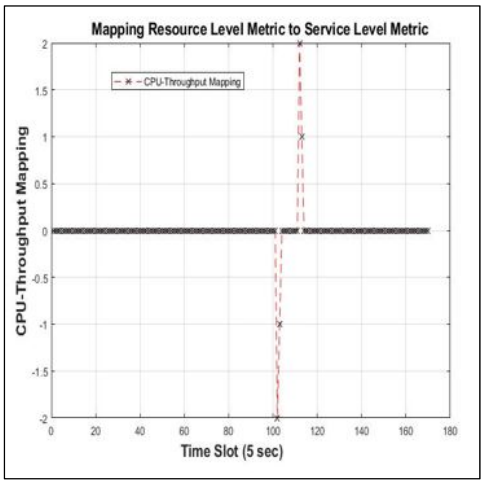

Figure 3.5 Config3 – Mapping

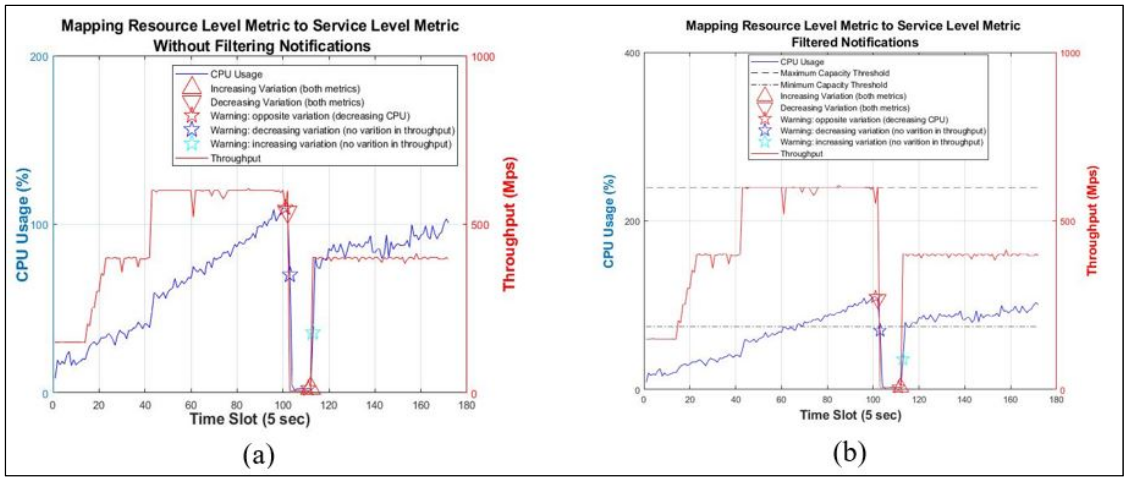

Figure 3.6 Config3 – Variation detection and filtered notifications

By processing Configuration 4 data, our proposed algorithm was able to detect multiple variation types. The mapping process shows multiple cases of unusual changes; for instance, at positions 1, 101, 102, 103, the corresponding mappings are 1, 2, -2, -1, respectively (see Figure 3.7.a). Further, the detected change types are mainly warning types (8, 6, 4 and 3 at positions 1, 539, 103, 388, 389) and increasing types (1, 7 at positions 101, 387, 503, 508) (see Figure 3.8.a). The types found reflect the large variation during the increase of CPU usage or the increase of workload, which are confirmed by the periodicity detection process. Indeed, the detected change type positions correspond to the detected large deviation positions (see Figure 3.7.b). Consequently, all the resource adaptation and warning notifications are filtered, as are the increase notification (type 7) at position 387 because the resource consumption is under the minimum capacity threshold (see Figure 3.8.b). We also notice that no variation type is detected between positions 150 and 300 despite the sharp decrease observed. This is probably due to the online process based on a sliding window, which subdivides data and affects the divergence evaluation and the outlier detection.

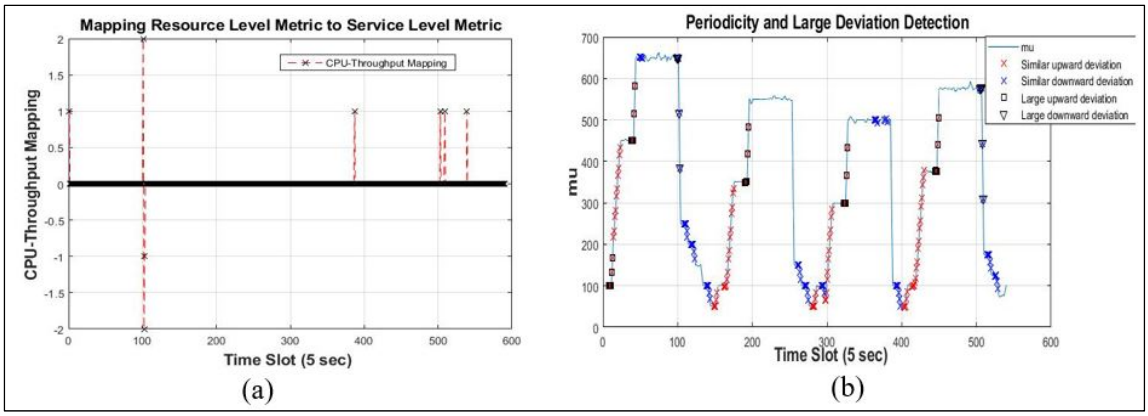

Figure 3.7 Config4 – Mapping and periodicity detection

The outlier detection for Configuration 5 also shows several outliers in two metrics. Their mapping results are -1 and -2 in the case of a sharp decrease and 1 and 2 in the case of a sharp increase at several times (see Figure 3.9.a). Moreover, these results are enhanced by the periodicity detection result which shows large deviations where outliers are detected (see Figure 3.9.b). For example, between positions 239 and 253, we observe the following mapping values: -1, 1 (between 241 and 248), 2 (251 and 253) and 0 (252) and their corresponding change types 5, 7, 1 and 4, respectively (see Figure 3.10.a). These results illustrate the ability of our algorithm to detect significant changes and their types (increase or decrease) when they occur. The algorithm is also able to detect such changes during the other cycles in Configuration 5, with the second cycle (between 105 and 159), seventh and eighth cycles (between 455 and 590) being the exceptions (see Figure 3.9.a and Figure 3.10.a). During the filtering process, all warnings and resource adaptation notifications are filtered. In addition, repeated change types are filtered as high priority notifications (e.g., change type 7 between positions 442 and 452), as presented in Figure 3.10.b.

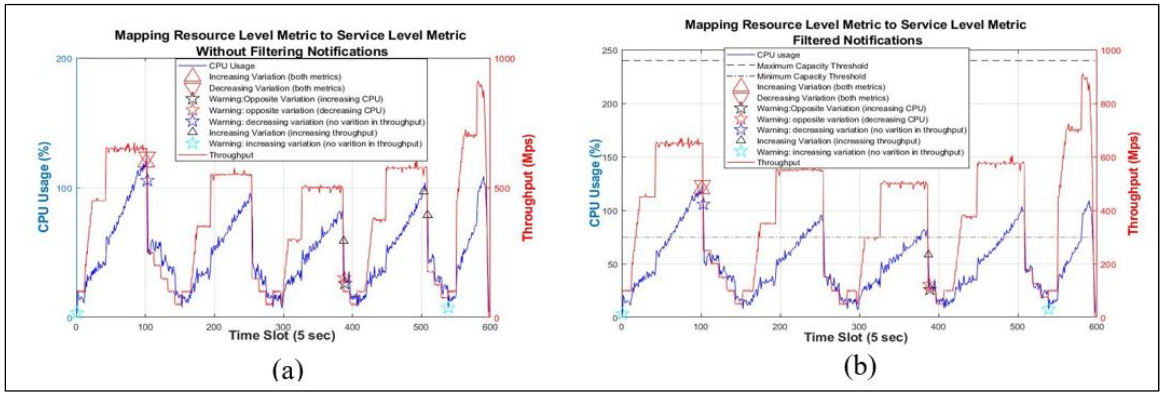

Figure 3.8 Config4 – Variation detection and filtered notifications

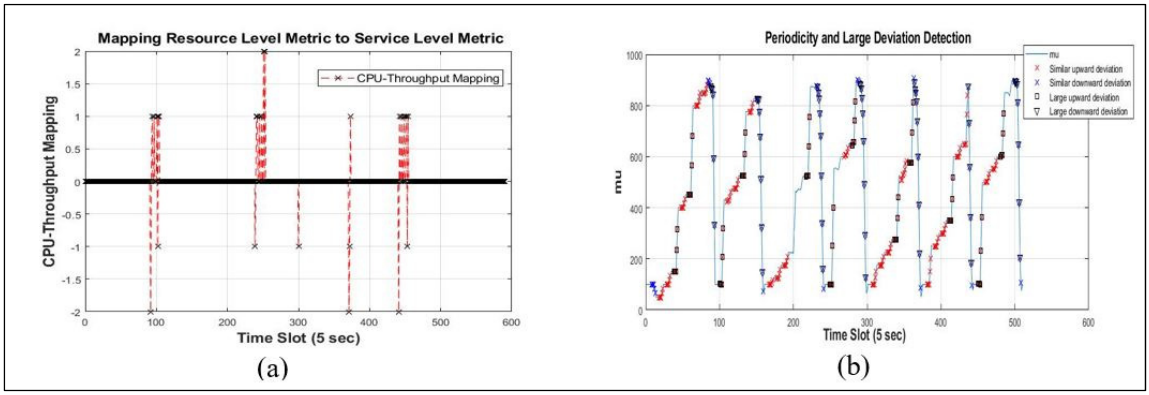

Figure 3.9 Config5 – Mapping and periodicity detection

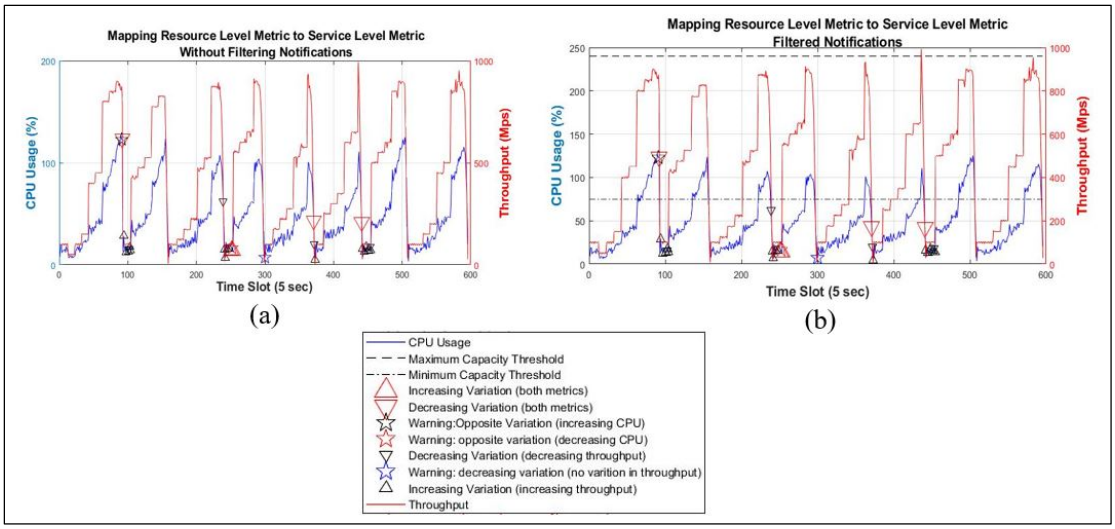

Figure 3.10 Config5 – Variation detection and filtered notifications

We further tested our algorithm in the case of fluctuant workloads, as observed in the data provided by the Ericsson Company. We also tested four types of metrics, namely, throughput (Dset3) and latency (Dset4) for the service level and CPU usage (Dset1) and memory usage (Dset2) for the resource level. The mapping between CPU usage and throughput (see Figure 3.11.a) shows multiple cases of workload increases (1) and decreases (-1) as observed, for instance, at positions 294, 298, 312, 316.

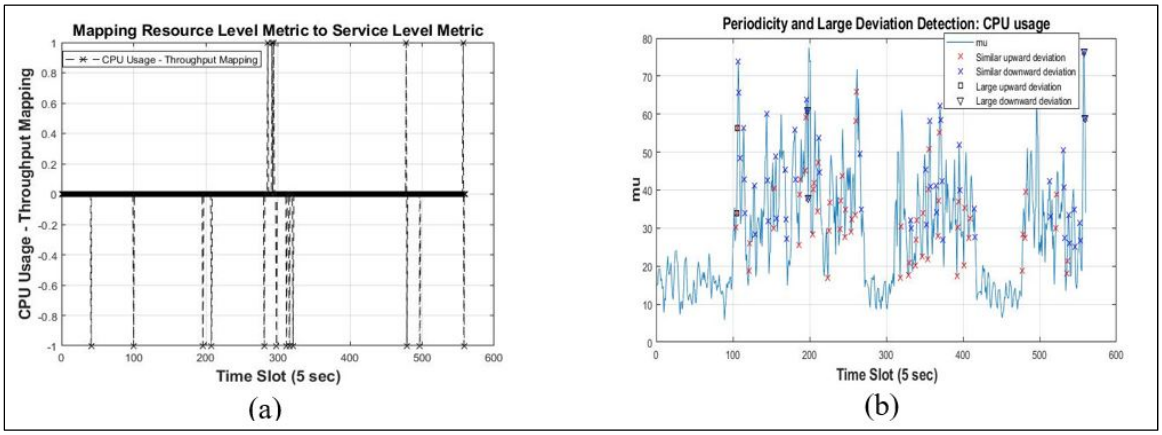

Figure 3.11 Dset1 and Dset3 (CPU usage and Throughput) - Mapping and periodicity detection

Furthermore, the periodicity process detects several fluctuations (see Figure 3.11.b), two upward deviations (positions 105, 106) and two large downward deviations (positions 197, 198) in CPU usage data, while the throughput data appear to be stable, except for large downward deviations detected at positions 40, 41, 42, and 98, 99, 100, due to sharp increases followed by sharp decreases. As well, the change types detected by our algorithm are mainly warning types 6 and 8 (for instance, at positions 196 and 286, respectively) and a decreasing type (5 at positions 41, 100) (see Figure 3.12.a). All notifications corresponding to detected change types are filtered (see Figure 3.12.b) because the CPU consumption was constantly under the minimum capacity threshold.

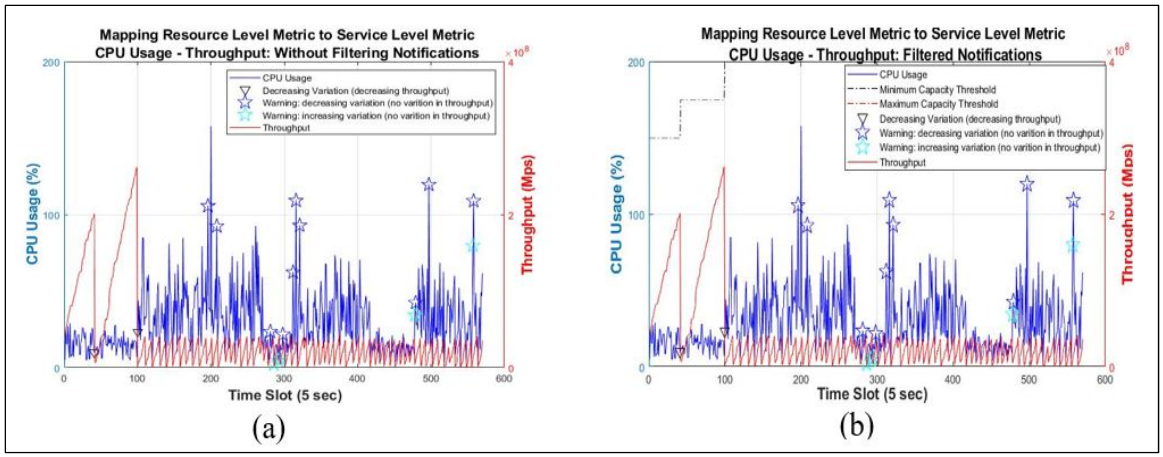

Figure 3.12 Dset1 and Dset3 (CPU usage and Throughput) - Variation detection and filtered notifications

On the other hand, the mapping between memory usage and latency presents several cases of unusual changes; for instance, at positions 1, 41, 442, 443, the corresponding mappings were 1, -1, 2, -2, respectively (see Figure 3.13.a). Next, our algorithm detects multiple phases of downward deviations corresponding to sharp decreases (e.g., positions 40, 41, 42) and one phase of upward deviations corresponding to sharp increase (positions 476, 477, 478) in memory usage (see Figure 3.13.b). Further, the increasing type 7 (increase in latency) is detected at several successive positions (e.g., between positions 1 and 9), as well as the increasing type 1 (increase in both metrics) at positions 441, 442, 521, 522 and 523. The decreasing types 5 (decrease in latency) and 2 (decrease in both metrics) are also detected, for

instance, at positions 41 and 524, respectively. Additionally, two notifications for warning type 8 (increasing in memory usage and no variation in latency) are generated at positions 478 and 479 (see Figure 3.14.a). Because, the memory usage is mostly greater than the maximum capacity threshold, the notifications generated by our algorithm are filtered, except for the notifications from positions 541 to 547, where the memory usage is between the maximum and the minimum capacity thresholds (see Figure 3.14.b).

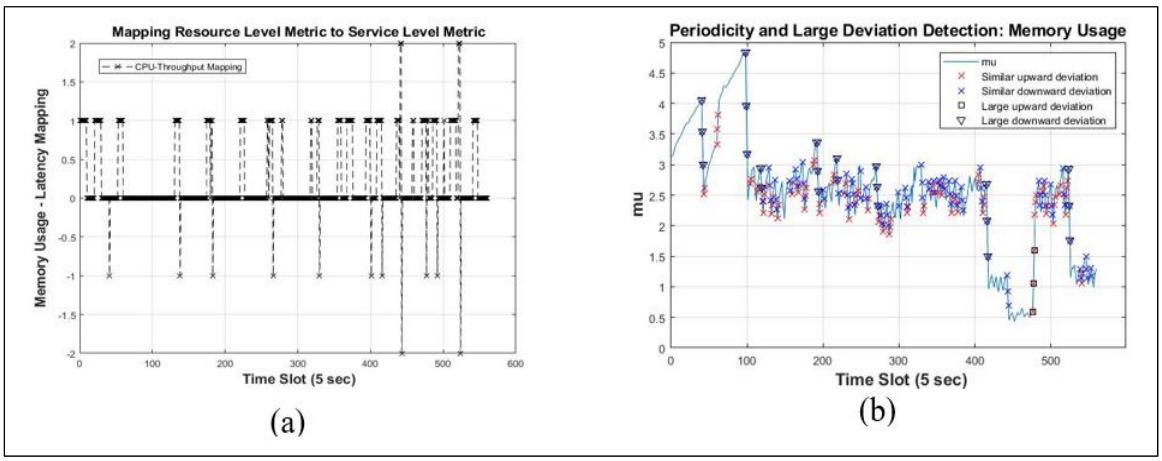

Figure 3.13 Dset2 and Dset4 (Memory usage and Latency) - Mapping and periodicity detection

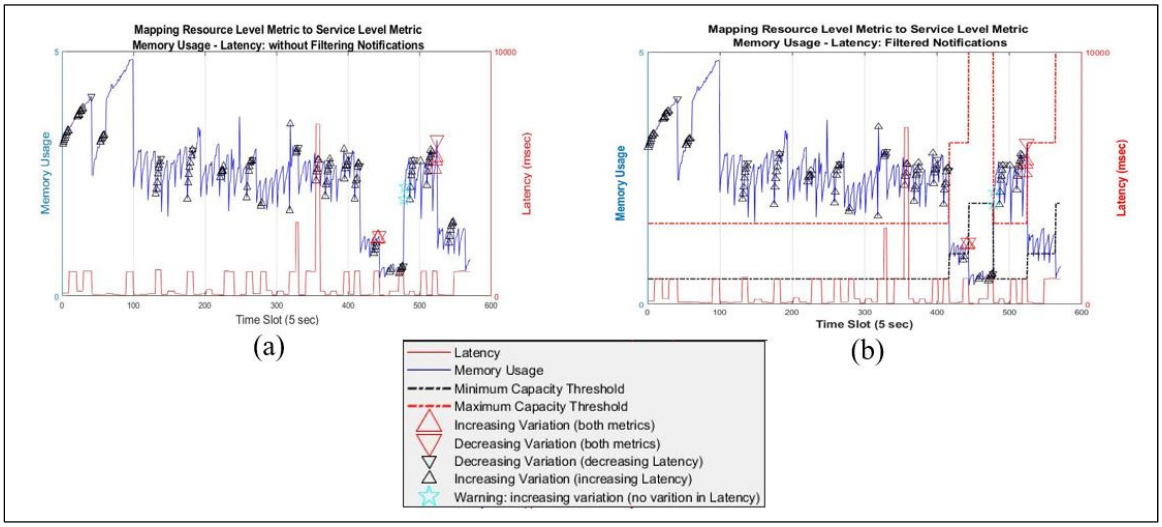

Figure 3.14 Dset2 and Dset4 (Memory usage and Latency) - Variation detection and filtered notifications

To summarize, these results prove that our algorithm is capable of detecting unusual changes, both in service-level and resource-level metrics, when they occur, as well as to map detected changes and to provide the types for such fluctuations. Afterward, we evaluate the accuracy of the proposed abnormal behavior detection algorithm (i.e., ability to correctly recognize the type of each detected abnormal behavior) using multi-class classification metrics (e.g., accuracy, precision, sensitivity).

Therefore, we first analyze data in Configurations 1, 2, 3, 4, 5 and Datasets 1, 2, 3, 4, and then we assign a type for each abnormal behavior to create a classification scheme which we can use to compare the algorithm results with the classes we provide. In the next step, we determine a confusion matrix for multi-class classification, and we compute the accuracy, precision and sensitivity for each change type (class). The results of the evaluation metrics for each configuration are presented in Table 3.8.

The accuracy of our abnormal behavior detection algorithm is 100% for Configurations 1 and 2, which means a complete conformity of our algorithm results with the actual classification. Yet, the accuracy in Configuration 3 is 99%, with change types 4 and 6, and 100% for the other types. The precision for change type 4 is 50% because at position 111, our algorithm has mistakenly identified the abnormal behavior as type 4, while in reality, it is type 6. Similarly, the sensitivity for type 6 is 50% because the change at position 111 is not recognized by our algorithm as a change type 6. In the other cases, the precision and the sensitivity metrics are both 100%, which reflect a very good performance for our proposed algorithm.

The accuracy of the detected change types in Configurations 4 and 5 varies between 95% and 100%. Nevertheless, the algorithm detection is less precise and sensitive, mainly in the case of warning types. For instance, types 4, 6 and 8 in Configuration 5 are detected, but they are misclassified, meaning that these types of variations are detected by our algorithm, but their positions are inconsistent with the classification scheme. In such cases, the classification errors are short-lived, i.e., occur at 1, 2 or 3 consecutive or close positions, and are possibly due to the time taken by the monitoring tools to react to workload variation at the service level (e.g.,

variation of the throughput) and its reaction at the resource level (e.g., variation of the CPU usage).

Table 3.8 Results of the evaluation of abnormal behavior detection algorithm for configurations 1, 2, 3, 4, 5 and datasets 1, 2, 3, 4

	Change Type	**Accuracy (%)**	**Precision (%)**	**Sensitivity (%)**	**Explanation**
Conf1	0	100	100	100	
	1-8	100	0	0	Not observed[a]
Conf2	0	100	100	100	
	5	100	100	100	
	1-4, 6-8	100	0	0	Not observed
Conf3	0	100	100	100	
	1,2,8	100	100	100	
	4	99	50	100	
	3,5,7	100	0	0	Not observed
Conf4	0	98	98	100	
	1	99	100	33	
	2,6	99	100	25	
	3,8	100	100	100	
	4,5	99	0	0	
	7	99	33	100	
Conf5	0	95	97	97	
	1	99	50	50	
	2,6	98	100	21	
	3	99	100	50	
	4,6,8	99	0	0	
	5	99	25	100	
	7	97	12	66	
Dset1 and Dset3	0	92	92	100	
	1,2	97	0	0	Not detected[b]
	3,4,7	99	0	0	Not detected
	5	99	100	66	
	6	98	100	58	
	8	98	100	45	
Dset2 and Dset4	0	90	95	92	
	1,2,8	100	100	100	
	3,4,6	100	0	0	Not observed
	5	98	100	50	
	7	92	72	88	

a-Not observed: The change type is not observed in configuration/dataset
b-Not detected: The change type is observed, but is not detected by our algorithm

On the other hand, the precision and the sensitivity of our algorithm are also affected by long-term classification errors. The latter occur when a significant variation such as a sharp

increase/decrease is not detected in its entirety, probably due to the precision errors of divergence and outlier detection techniques (Kullback-Leibler divergence, Modified Z-Score) and their ability to detect abnormal variations among fluctuant data. Further, we notice that the sliding window size impacts the accuracy of the detection techniques used, which are based on probability distributions and median of data.

The results using data from the Ericsson Company show the ability of our algorithm to detect unusual variations in cases of fluctuant workloads. Indeed, the accuracy of the change type detection varies between 92 and 99% for CPU usage and throughput. However, the precision and the sensitivity vary between 0 and 100% and between 0 and 66%, respectively, because of certain undetected change types. For instance, the increasing variation in both metrics corresponding to change type 1 was not detected as we can see at position 139, while the CPU usage increases from 5.66% to 45.32% and the throughput increases from 6154 to 2.26 10^6 MPS. On the other hand, for memory usage and latency data, we note that the change types 1, 2, 8 are correctly detected, with 100% accuracy, precision and sensitivity. Further, the change types 3, 4 and 6 (warning change types) are not observed, and our algorithm does not detect them (100% accurate). The decreasing and increasing variations (change types 5 and 7, respectively) are also detected, with an accuracy greater than 92%, but the sensitivity varies from 50% to 88% because of variations that were not reported by our algorithm (false negatives).

To summarize, these results show that our algorithm is able to detect and accurately provide notification regarding unusual variations that denote possible abnormal behavior in various types of workloads and systems, but classification errors (short-lived or long-term errors) impact the sensitivity and the precision in certain cases of variations because of the reaction time, the precision errors of outlier detection techniques, and the sliding window size.

3.9.3.1 Execution Time

After the evaluation of the proposed algorithm accuracy to detect and provide notifications regarding abnormal changes, we will now verify whether the generated notifications are on time. Indeed, if unusual behavior is observed, our algorithm has to respond quickly from the time the notification is issued to the time an action is taken. Therefore, we compute the execution time of our algorithm for every dataset to deduce the window of opportunity in seconds for the system to respond and take action.

For every sliding window, we compute the execution time of abnormal behavior detection techniques, namely, Kullback-Leibler divergence, Modified Z-Score, periodicity detection, occurrence computing, mapping and notification (Algorithm 3.1: Line 6-17). The abnormal behavior detection execution time was in milliseconds, except for when large deviations (sharp decrease or sharp increase) are detected because of the processing of historical data by the periodicity algorithm. We can observe in Figure 3.15 that the larger the historical data size, the slower the execution time. The latter was 5 and 10 sec for Config2 and Config3, respectively, and 26 and 29 sec for Config5 and Config4, respectively. Further, the execution time of our algorithm using data from the Ericsson Company was 30 and 31 sec for memory usage-to-latency mapping and CPU usage-to-throughput mapping, respectively. Furthermore, the average execution time of several configurations was below 3 seconds (see Figure 3.16), and below 13 seconds for datasets provided by Ericsson Company.

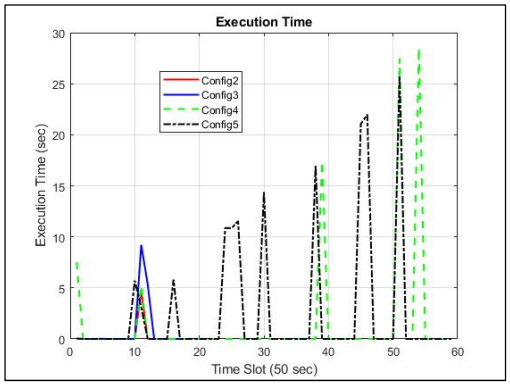

Figure 3.15 Execution Time: Config. 2-5

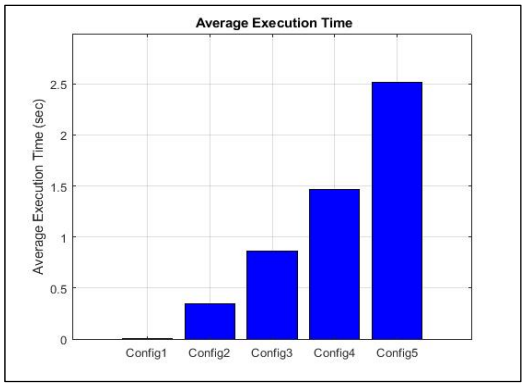

Figure 3.16 Average Execution Time:
Config. 1-5

For all configurations and datasets, we notice that the obtained execution time remains smaller than 50 seconds, which is the sliding window duration (m=10). However, we can propose to reduce the periodicity detection runtime step by resetting the historical data at a given number of sliding windows.

3.10 Conclusion and future work

Fluctuations in resource demand or unexpected and abnormal variations in system behavior lead to critical service performance degradation or service failure and to costly SLA violations. Existing approaches lack genericity and efficiency because they are mainly based on specific and excessive allocation of resources, which involve resource loss. They also propose monitoring systems without validation of detected variations mapping of their states at both the service and resource levels, and without detecting patterns. In this work, we propose a generic, dynamic, and accurate abnormal behavior detection in virtualized systems. The proposal aims to minimize the system reaction time when unusual changes in workload occur by providing a continuous, fast and precise evaluation of the system state and the action to be undertaken. This can lead to minimization of possible SLA violations and loss of resources, without any prior knowledge of the system or any assumptions on its behavior. Several experiments were conducted using several metrics and various types of workload profiles. The

results show that our algorithm is able to detect and map unusual variations, in both service- and resource-level metrics, as well as the periodicity of detected variations. Moreover, the algorithm can provide the types of these abnormal variations and generate corresponding adaptation resource notifications or warnings with an accuracy ranging from 92% to 100%. To improve the precision and sensitivity of the detected variation types, we plan to test other outlier detection techniques, for instance, the Mahalanobis distance, and include logs gathered from monitoring tools in order to corroborate notifications and warnings provided by our algorithm. Moreover, we propose to further analyze the impact of sliding window size on the performance of the algorithm and to test a new approach for a dynamic adjustment of the sliding window size.

Acknowledgment

This work has been supported in part by Natural Science and Engineering Research Council of Canada (NSERC), in part by Ericsson Canada and in part by Rogers Communication Canada.

CHAPITRE 4

ADVANCED WORKLOAD MODELING FOR CLOUD SYSTEMS

Cédric St-Onge [a], Souhila Benmakrelouf [b], Nadjia Kara [c], Hanine Tout [d], Claes Edstrom [e], Rafi Rabipour [f]

[a, b, c, d] Department of software engineering and IT, École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[e, f] Ericsson Canada,
8275 route Transcanadienne, Saint-Laurent, Québec, Canada, H4S 0B6

Paper accepted for publication in "Journal of Cloud Computing",
September 2020

Dans ce chapitre, nous présentons le travail de recherche effectué dans le cadre de cette thèse et qui a été accepté pour publication dans (St-Onge, Benmakrelouf, et al., 2020). Dans ce travail, nous avons développé une nouvelle approche pour estimer la charge de trafic qui combine un filtre de Kalman (Hu et al., 2013) et la régression à vecteurs de support (SVR-Support Vector regression) (Cortes & Vapnik, 1995). Cette approche est comparée à celle développée par le co-auteur Cédric St-Onge et laquelle combine l'approche Hull-White et l'algorithme génétique.

4.1 Abstract

Workload models are typically built based on user and application behavior in a system, limiting them to specific domains. Undoubtedly, such practice forms a dilemma in a cloud environment where a wide range of heterogeneous applications are running, and many users have access to these resources. The workload model in such infrastructure must adapt to the evolution in the system configuration parameters like job load fluctuation. The aim of this work is to generate generic workload models which are independent of user behavior and the applications running in the system, and that are able to fit any workload domain and type, able to model sharp workload variations that are most likely to appear in cloud environments, and

with high degree of fidelity with respect to the observed data in a short period of execution time. In this work we propose two approaches for workload estimation. The first approach is a combination of Hull-White and Genetic Algorithm (GA), while the second one is a combination of Support Vector Regression (SVR) and Kalman-filter. Thorough experiments are conducted on real datasets to study the efficiency of both propositions. The results show higher accuracy for the Hull-White-GA approach with marginal overhead over SVR-Kalman-Filter combination.

Keywords: cloud computing, workload modeling, workload estimation, Hull-White model, genetics, support vector regression, Kalman filter.

4.2 Introduction

With the ubiquity of Cloud Computing technologies over the last decade, Cloud providers and researchers have strived to design tools for evaluating and enhancing different Quality of Service (QoS) aspects of their systems, mainly performance, availability and reliability. Failing to comply to such aspects can compromise service availability and incur Service Level Agreement (SLA) violations, and hence impose penalties on cloud providers. The development of system management policies that support QoS is therefore crucial. However, the latter is a challenging task, as it must rely on evaluation tools which are able to accurately represent the behavior of multiple attributes (e.g., CPU, RAM, network traffic) of Cloud systems. It is also complicated by the essence of Cloud systems, built on heterogeneous physical infrastructure and experiencing varying demand; these systems have different physical resources and network configurations and different software stacks. Further, the reproduction of conditions under which the system management policies are evaluated, and the control of evaluation conditions are challenging tasks.

In this context, workload modeling facilitates performance evaluation and simulation as a “black box” system. Workload models allow Cloud providers to evaluate and simulate resource management policies aiming to enhance their system QoS before their deployment in full-scale

production environments. As for researchers, it provides a controlled input, allowing workload adjustments to fit particular situations, repetition of evaluation conditions and inclusion of additional features. Furthermore, workload simulation based on realistic scenarios enables the generation of tracelogs, scarce in Cloud environments because of business and confidentiality concerns. Workload modeling and workload generation are challenging, especially in Cloud systems, due to multiple factors: workloads are composed of various tasks and events submitted at any time, heterogeneous hardware in a Cloud infrastructure impacts task execution time and arrival time and the virtualization layer of the Cloud infrastructure incurs overhead due to additional I/O processing and communications with the Virtual Machine Monitor (VMM). These factors make it difficult to design workload models and workload generators fitting different workload types and attributes. In the current state of the art, effort is instead deployed to design specialized workload modeling techniques focusing mainly on specific user profiles, application types or workload attributes.

To tackle the above issues, we propose in this article a hybrid workload modeling and optimization approach to accurately estimate any attribute of workload, from any domain. The objective is to develop realistic CPU workload profiles for different virtualized telecom and IT systems, based on data obtained from real systems. Our proposition consists first of modeling workload data sets by using different Hull-White modeling processes, and then determining an optimal estimated workload solution based on a custom Genetic Algorithm (GA). We also propose a combination of Kalman filter (Hu et al., 2013) and support vector regression (SVR) (Cortes & Vapnik, 1995) to estimate workloads. Our IMS workload data sets include three variations of the same load profile, where CPU workload is generated by stressing a virtualized IMS environment with varying amounts of calls per second, thus producing sharp increases and decreases in CPU workload over long periods of time. Our IT workload types take two different approaches. The Google CPU workload under evaluation, for example, is composed of sharp spikes of CPU workload variations over short periods of time. By contrast, another CPU workload, namely Web App, is characterized by periodicity trends. These datasets therefore display unique trends and behavior that provide interesting scenarios to evaluate the efficiency of the workload modeling and workload generation techniques that we are proposing

in this paper. The evaluation of the mean absolute percentage error (MAPE) of the best estimated data provided by our Hull-White based approach against the observed data, shows significant improvement in the accuracy level compared to other workload modeling approaches such as standard SVR and the SVR with a Kalman filter.

The main contributions of this work consist of:

1. Proposing a generic workload modeling approach fitting any workload domain (e.g., Telecom, IT) and any workload attribute (e.g., CPU, RAM, I/O);
2. Providing an automated workload generating tool capable of generating estimated workload with minimal user input;
3. Generating workload models without requiring knowledge of the inner behaviors of the modeled systems (i.e., "black box" approach);
4. Generating workload profile data while limiting dependence on external organizations for providing such data.

4.3 Background

With the rise of Cloud computing in the last decade, there has been an increasing amount of research conducted to help Cloud providers improve their systems performance, e.g. energy efficiency and Quality of Service (QoS). To achieve this goal, a common practice is to evaluate a system's workload. The notion of predictable workload denotes a "normal" and continuous usage of system resources. A non-predictable workload, on the other hand, is one which is triggered by a short, unexpected and extreme event. It can be, for example, a surge in calls being processed by an IMS system during an earthquake. In the context of Cloud computing, this is an important factor to consider since it has impact on how to handle resource allocation. In Shaw & Singh (Shaw & Singh, 2015), the authors propose different methodologies and allocation decisions based on both workload types. In the presence of predictable workload, for example, their algorithm is designed to invoke a virtual machine (VM) migration only if a fixed threshold is violated for a sustained time. In the presence of non-predictable workload, on the other hand, VM migration takes place when utilization is above a fixed threshold value.

The use of predictable workload data is therefore preferred for building workload models where a user wishes to forecast future load, while non-predictable workload data is preferred for building workload models where there are variations such as sharp, critical spikes in the load that might impact the system.

Workload modeling appears in many contexts and therefore has many different types (Feitelson, 2015). From the outset, it is therefore essential for someone planning to evaluate specific workloads to understand what domains, types and attributes of workloads is necessary to investigate. It is also important to study if the workload is taking the shape of a static or dynamic rate of events, and whether the workload is predictable or non-predictable. The following will explore all these concepts in the context of our research.

4.4 Related Work

In this section, we review existing works relevant to workload modeling to set the perspective for our contributions. The present review emphasizes on, but is not limited to, virtualized Cloud environments such as Google real traces and Open IMS datasets. This overview aims to give a broader picture of general types of workloads and workload modeling techniques that have been previously addressed by the research community

4.4.1 Workload Domains

Workload domain characterization is the first step and the uppermost level that should be considered before planning performance evaluation. It has a major impact upon what type of workload is to be considered. Domains vary in shape and scope, depending on the size and purpose of the environment; one can go from the performance evaluation of a single application on a workstation, to a full-scale performance evaluation of a multi-tenant, heterogeneous cloud environment (e.g., Amazon EC2). Google clusters and virtualized IP Multimedia Subsystem (IMS) cloud environments, both subjects of this work, are considered

as workload domains belonging to the field of Cloud computing. Different relevant works have been cited in this context.

Moreno et al. (Moreno, Garraghan, Townend, & Jie, 2013) provide a reusable approach for characterizing Cloud workloads through large-scale analysis of real- world Cloud data and trace logs from Google clusters. In this work, we address the same workload domain. There are, however, differences in their approach, starting with the dynamic behavior of their workload, in contrast with the static behavior of ours. The difference between a static and dynamic workload behavior is discussed next. MapReduce and Hadoop performance optimization (Moreno et al., 2013), (H. Yang, Luan, Li, & Qian, 2012), (An, Zhou, Liu, & Geihs, 2016) also bring interesting avenues because of the nature of the data intensive computing taking place in such environments, and because this framework is at the core of most of the leading tech company datacenters in the world, like Google, Yahoo and Facebook. This type of optimization, on the other hand, focuses on long-term analysis of predictable workloads and scheduled tasks, rather than quick bursts in demand for a specific application in a non-predictable manner. Performance evaluation of web and cloud applications (An et al., 2016), (Bahga & Madisetti, 2011), (Magalhães, Calheiros, Buyya, & Gomes, 2015)is another popular domain worthy of consideration. This domain usually involves the evaluation of user behavior, among other things, which is less prevalent in other domains. However, this domain is typically application-centric and rarely considers the workload characteristics of the whole cloud environment.

4.4.2 Workload Types

The next step when planning performance evaluation is to define what workload types to investigate. Feitelson (Feitelson, 2015) gives a good description of the constitution of a workload type. For instance, the basic level of detail of a workload is considered an instruction, many of which compose a process or a task. A set of processes or tasks can in turn be initiated by a job sent by a user, an application, the operating system, or a mix thereof.

In the field of cloud computing, the purpose of performance evaluation is mainly to optimize hardware resource utilization. There is therefore little to no variation in workload types, aside from minor differences in philosophy or context from the researchers. Workloads typically originate from a mix of user, application and task workload types. For instance, the workloads studied by Magalhaes et al. (Magalhães et al., 2015) and Bahga et al. (Bahga & Madisetti, 2011) are spawned from the behavioral patterns of specific user profiles using select web applications. A similar work by Moreno et al. (Moreno et al., 2013) uses a similar approach but focuses on the workload processed within a cloud datacenter driven by users and tasks. Studying these workload types can be useful in evaluating our own datasets, since the latter also depend on user behavior. In fact, user behavior is the prime factor in workload generation for our performance evaluations. Another approach proposed by An et al. (An et al., 2016) consists of viewing the user, the application and the service workload types as three layers of granularity. The users launch a varying amount of application-layer jobs, which in turn execute a varying number of service-layer tasks. This method is effective in the context of predictable, dynamic workloads. In some cases, only one workload type will be investigated, such as parallel processes and jobs under a specific cloud environment (H. Yang et al., 2012). Such work is intended to acquire very specific benchmarks from selected applications like MapReduce, and to evaluate optimal hardware configuration parameters in a cluster.

4.4.3 Workload Attributes

Workload attributes are the last step to consider when planning the performance evaluation of a system. Attributes are what characterize workload types, and directly influence hardware resources of the system. It is therefore essential to have a clear idea of how each instruction interacts with one or more resources from the system if you want to correctly interpret the workload data. For instance, I/O (disk and memory usage, network communications, etc.) attributes include the distribution of I/O sizes, patterns of file access and the use of read and/or write operations (Feitelson, 2015).

In this work, only the scheduling of the CPU is of interest. Hence, the relevant attributes are each job's arrival and running times (Feitelson, 2015). CPU scheduling is a common interest in performance evaluation, and most studies analyze many workload attributes related to this resource. In An et al. work (An et al., 2016), the system CPU rate, threads, Java Virtual Machine (JVM) memory usage and system memory usage are analyzed. Similar attributes are considered in Magalhaes et al. work (Magalhães et al., 2015), where the system CPU rate, memory rate and the users' transactions per second are evaluated. Other works are much more thorough in their analysis (Moreno et al., 2013), where attributes are subdivided in user patterns (submission rate, CPU estimation, memory estimation) and task patterns (length, CPU utilization, Memory utilization). Putting aside the differences in workload domains such as IMS and Google performance evaluation, we can safely assume that the system CPU behavior remains the same in both cloud environments.

4.4.4 Static and Dynamic Workloads

As Feitelson describes it: "An important difference between workload types is their rate of events. A static workload is one in which a certain amount of work is given, and when it is done that is it. A dynamic workload, in contrast, is one in which work continues to arrive all the time; it is never done." (Feitelson, 2015). Evaluation of static versus dynamic workloads will depend on the objective to be achieved. For instance, since jobs from a static workload are processed by a "clean" system, it usually implies the need to evaluate a system's behavior over a short time span. Jobs processed in a dynamic manner, on the other hand, require the evaluation of a system's behavior over a longer time span (several hours, days), since the same set of jobs are being processed by a system that is continually processing other jobs.

Static workload evaluation usually involves the analysis of a smaller number of workload types and attributes, increasing the complexity of the process. Since job execution times are not considered, it incurs a "drift" in variation in the rate of usage resources, since the system keeps processing jobs stacked in its queue while new jobs arrive. Dynamic workload evaluation overcomes this issue by adding a probabilistic and/or distributed (e.g., normal distribution,

exponential distribution) approach to job arrivals and execution times that give a more accurate representation of the impact of workload types and attributes on the system resources (An et al., 2016), (Bahga & Madisetti, 2011), (Magalhães et al., 2015), (H. Yang et al., 2012).

4.4.5 Workload Modeling

The goal of workload modeling is to be able to create workloads that can be used in performance evaluations. Of course, the objective is to get a workload model as close as possible to the real workload: "Workload modeling is the attempt to create a simple and general model, which can then be used to generate synthetic workloads at will, possibly with slight (but well-controlled!) modifications" (Feitelson, 2015). It always starts with measured workload data and is a common alternative to using the traced workload directly to drive simulations.

To achieve the main objective of this work, which is to design realistic workload profiles for different virtualized telecom and IT systems, and also to provision cloud environment resources so as to fulfill service level agreements (SLA) as a use case of this work, we use workload modeling for a special case of performance evaluation; capacity planning. In a sense, capacity planning is performance evaluation in reverse. In other words, instead of deriving the performance of a given system configuration, we seek the configuration that will provide the desired performance (Menasce, Almeida, Dowdy, & Dowdy, 2004). The most common approach found in the literature to create a workload model fitting our needs is to create a statistical summary of an observed workload. We apply this summarization to all the workload attributes (e.g., CPU, memory usage, I/O, etc.), then we fit distributions to the observed values of the different parameters (Moreno et al., 2013), (Bahga & Madisetti, 2011), (Magalhães et al., 2015). In one case, for example, a statistical analysis of the user requests is performed to identify the right distributions that can be used to model the workload model attributes such as inter-session interval, think time and session length (Bahga & Madisetti, 2011). Four candidate distributions are then considered for each workload attribute based on efficiency: exponential distribution for inter-session arrival, hyper-exponential distribution for modeling think times

and inter-session intervals, Weibull distribution and Pareto distribution to model session lengths.

Other approaches applied to entirely different fields can also be of great interest. For instance, Tahmasbi and Hashemi (Tahmasbi & Hashemi, 2014) propose a model for forecasting urban traffic volume by using, among other things, the Hull-White model. This model is almost exclusively used in the finance sector to predict fluctuations in stock prices over a short period of time. In the case of short-term urban volume, the Hull-White model provided interesting results and we expect it does the same for our needs. Also noteworthy is the use of quadratic splines with irregular intervals as statistical summaries, and the use of the Wiener process as a distribution fit.

Other existing works have proposed workload prediction using linear regression (Lloyd et al., 2013), Neural Networks (Islam et al., 2012), Kalman filter (Zhang-Jian et al., 2014), (W. Wang et al., 2012) and support vector regression (SVR) (Z. Wei et al., 2013). An optimal strategy for resource management should allow dynamic on-demand adjustment and provisioning of resources. This can be greatly facilitated if the workload can be predicted accurately. In this context, Hu et al. (Hu et al., 2013), have proposed to address CPU usage estimation based on time series. Their approach is based on support vector regression and Kalman filter, particularly smooth Kalman filter, in order to remove the noise in the data, and reduce the prediction error rate. Higher weights are assigned to the latest data in the training set as such data provide more recent information on the behavior of the system. This idea was developed previously by Cao et al. (Cao & Tay, 2003), in their study of SVR with adaptive parameters in the prediction of time series in the financial domain. These approaches have outperformed prediction algorithms which are based on linear regression and neural networks, and even standard SVR.

4.4.6 Proposed Approach Motivation

Authentic industrial-grade workload data is sparse and often incomplete. Also, actual workload models are limited to very specific domains and are not flexible enough, as they cannot to be applied to a combination of domains such as IT and Telecom Virtualized Cloud environments. Further, the workload model must adapt to evolution in the system configuration parameters (e.g., Load, switched off CPU core). In this context, our proposed approach differs from other existing solutions in different aspects:

- combining Hull-White processes with GA automates the selection of segmented potential solutions through different models;
- GA reinforces our Hull-White process and optimizes our workload models by selecting the fittest solutions through many generations;
- significant improvement in execution time compared to other types of workload estimation models (e.g. auto-regression, moving average);
- accommodation of on-demand workload profile changes in a simulation thanks to adaptable and continuous spline functions.

4.5 Problem Illustration

One of the many challenges that arise when aiming to evaluate Cloud system policies, is the ability to get reliable data and tracelogs from organizations hosting Cloud environments. To circumvent this issue, workload modeling is introduced as a good alternative, enabling research teams to generate large amounts of workload profile data for use in the course of their work, while limiting dependence on external organizations for providing such data.

Many workload modeling solutions exist in the research community, but none of them answers the needs for a general purpose, generic modeling algorithm that can fit different types of telecom and IT workloads. Current workload models are limited as they are built on top of user and application behaviors of specific domains. While this might sound adequate for a single web server where the number and the type of applications are limited, this is not the case in a

cloud environment where a wider range of heterogeneous applications are running, and many users have access to these resources. The workload model must adapt to the evolution in the system configuration parameters (e.g., booting up a new VM or enabling new CPU cores to handle exceeding levels of resource loads).

Such issues motivate the need to generate generic workload models:

- without having to know the specifics of user behavior and the deployed applications;
- with high degree of fidelity with respect to the observed data;
- able to efficiently estimate rapid workload variations and sudden/unpredictable changes.

4.6 Workload Generator Architecture

In this section, we give an overview of the proposed scheme for our workload generator. The novelty of our approach lies on the relationship between the Hull-White workload modeling process and a custom Genetic Algorithm (GA) involved in the workload generation and optimization process, as outlined in Figure 4.1. For Hull-White modeling, we estimate first the mean and the standard deviation of an observed workload data set in a chronological order. To get smooth and continuous functions of mean and standard deviation, our workload generator employs uniform and non-uniform quadratic spline curves. Next, fixed and variable theta values (θ, a Hull-White parameter) will be estimated. The full process of Hull-White modeling is explained in Section 4.7. The breakdown of segments for non-uniform splines and also for variable thetas are evaluated in an entirely automated process of the workload generator. The latter sets boundaries where there are large variations in the values of the mean and/or standard deviation over a short period of time. We thus obtain four different Hull-White models from which to generate our workload data, namely; uniform splines and fixed theta, uniform splines and variable theta, non-uniform splines and fixed theta and non-uniform splines with variable theta.

On its own, the Hull-White modeling process proves to be an efficient algorithm to estimate workload data. However, by enhancing the Hull-White modeling with a custom GA

optimization, our experiments demonstrate that the Mean Absolute Percentage Error (MAPE) of the estimated data generated by our approach is significantly improved, with minimal impact on the execution time. That proves to be especially true when pairing the benefits of the GA with the four afore-mentioned Hull-White models, each one providing different levels of fidelity, and excelling in different areas.

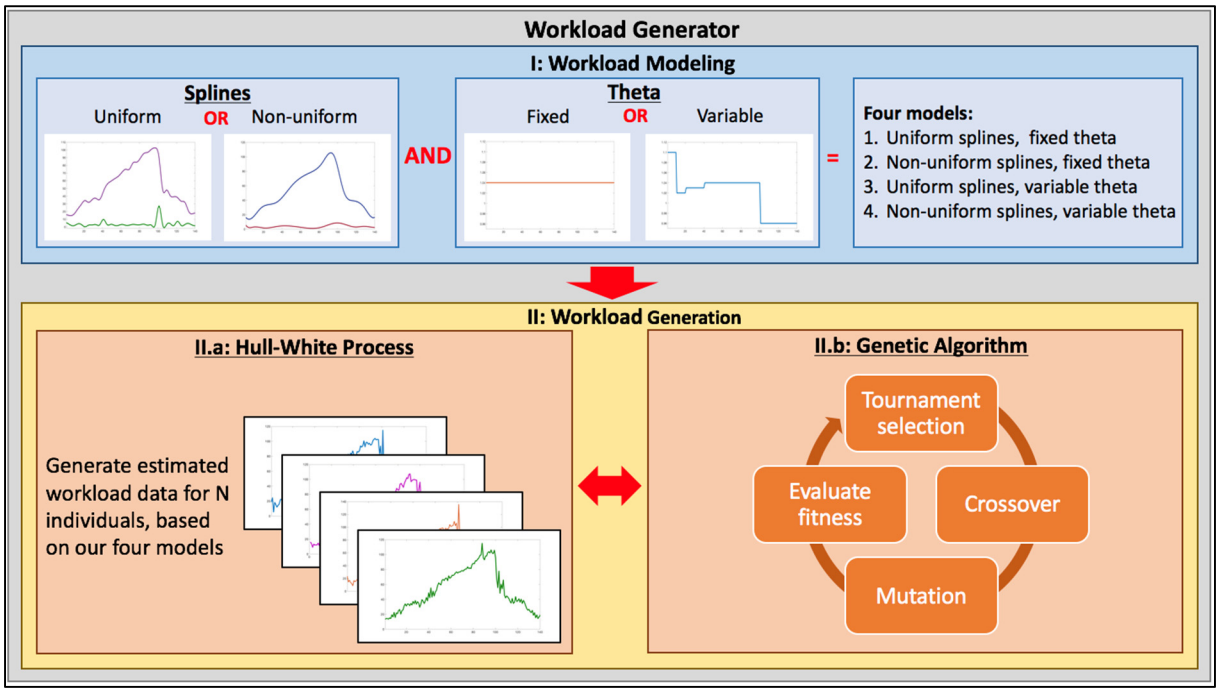

Figure 4.1 Proposed Workload Generator Scheme

To briefly summarize the necessary steps to obtain estimated workload data through this workload modeling approach:

1. We provide discrete observed workload data as an input to the workload generator;
2. The workload data is processed and four Hull-White workload models are automatically generated and saved under a workload profile;
3. A workload profile is used in full or in part of a workload generation process;
4. The custom GA generates estimated workload data from many instances of the corresponding workload profile(s) and proposes the fittest solutions.

4.7 Hull-White Workload Modeling

In this section, we present the formulation of the workload modeling problem and its underlying processes. The main objective of this process is to develop realistic workload profiles for different virtualized telecom and IT systems, based on data obtained from real systems, without requiring knowledge of their inner working processes (Black-box approach).

4.7.1 Stochastic Differential Equations (SDE)

From each consecutive samples of the real data, we generate mean and standard deviation values. SDEs are an excellent choice to model the time evolution of dynamic systems subject to random changes.

$$dX_t = \mu(t)dt + \sigma(t)dW_t, \qquad t \geq 0. \tag{4.1}$$

where,

X_t : Observed workload

$\mu(t)$: Mean value of observed workload at time t

$\sigma(t)$: Variance value of observed workload at time t

W_t : Weiner process

4.7.2 Splines

Next, we generate splines for curve-fitting continuous mean $\mu(t)$ and continuous standard deviation $\sigma(t)$ values. Splines are used to estimate the mean $\mu(t)$ and standard deviation $\sigma(t)$ of a set of workload data in order to achieve smooth and continuous functions. We use both uniform and non-uniform splines. The first one uses knots (aka “anchor points”) set at regular time intervals (ex: one knot every 20 seconds), while non-uniform splines use knots set at irregular time intervals.

$$\hat{f}(X_t(t)) = \begin{cases} a_i t_{i-1}^2 + b_i t_{i-1} + c_{i-1} = \hat{f}(X_{i-1}) \\ a_i t_i^2 + b_i t_i + c_i = \hat{f}(X_i) \\ a_i t_{i+1}^2 + b_i t_{i+1} + c_{i+1} = 0 \end{cases} \quad i = 1, \dots, n \tag{4.2}$$

where,

$a_1 = 0$: First linear spline

$\hat{f}\big(X_t(t)\big) = \mu(t)$: Continuous mean

or

$\hat{f}\big(X_t(t)\big) = \sigma(t)$: Continuous standard deviation

4.7.3 Hull-White Process

The following describes the main properties of the Hull-White model, which is a popular choice of SDE in the finance sector for modeling future interest rates.

$$dX_t = (\mu(t) - \theta X_t)dt + \sigma(t)dW_t, \qquad t \geq 0. \tag{4.3}$$

where,

X_t : Observed workload

$\mu(t)$: Mean spline value of observed workload at time t

$\sigma(t)$: Standard deviation spline value of observed workload at time t

θ : Estimated parameter

W_t : Weiner process

4.7.4 Estimation of θ

The parameter theta θ is a key feature of the Hull-White process. It is an estimated value that provides a drift (an upwards/downwards motion) to the estimated workload. In this work, we generate models with both, fixed and variable theta values. For fixed theta, we take the whole

data from the data set and we calculate a single theta value. As for variable theta, we use time windows whose lengths depend on the variations in the data.

$$L_t(\theta) = \exp(-\int_0^T \hat{u}(t, X_T)dX_T - \frac{1}{2}\int_0^T \hat{u}^2(t, X_T)dt) \quad (4.4)$$

where,

$$\hat{u}(t, X_T) = \frac{\mu(t) - \theta X_T}{\sigma(t)} \quad (4.5)$$

Therefore, the Maximum Likelihood Estimation (MLE) is defined as

$$\widehat{\theta_T} = \arg max\ L_T(\theta) \quad (4.6)$$

4.8 GA Workload Estimation

A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection. It belongs to the larger class of evolutionary algorithms (EAs) and is commonly used in computer science to generate optimized solutions to complex search problems. To achieve this goal, a GA relies on bio-inspired operators such as mutation, crossover and selection to simulate the propagation of fittest individuals over consecutive generations.

4.8.1 Individuals

In GA, a population is a set of *n* individuals that form potential solutions. For workload estimation we define each individual/chromosome as a set of decimal values. Each chromosome is an estimated workload value for the current workload attribute of the workload profile under evaluation. For instance, if we evaluate the CPU attributes of an IMS workload profile, each individual would be composed of chronologically and randomly estimated CPU workload values provided by any of the defined Hull-White models (Figure 4.2).

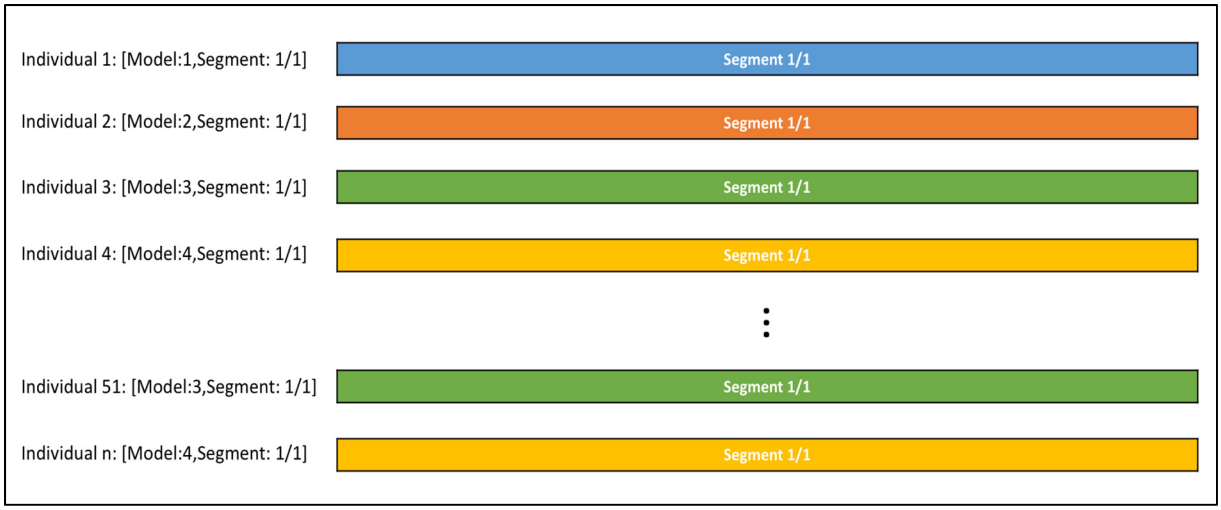

Figure 4.2 Individuals and Segments

4.8.2 Segments

An individual is divided into smaller segments, or genes, to go further in an evolution process of crossover and mutation operations. Each segment is of varying size and represents a portion of the individual (Figure 4.2). For instance, an individual of length L=100 can be divided into 4 even parts, hence creating 4 segments: segment 1 contains genes 1 through 25, segment 2 contains genes 26 through 50, etc.

4.8.3 Fitness Function

In GA, a fitness function is required to rank individuals in the current population. The score of an individual depends on how close its estimated values are to the observed values of a workload profile. In the problem at hand, we evaluate the fitness of an individual by its mean absolute percentage error (MAPE) score:

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{X_t - \hat{X}_t}{X_t}\right| \tag{4.7}$$

where,

X_t : Observed value

$\hat{X}_t$: Estimated value

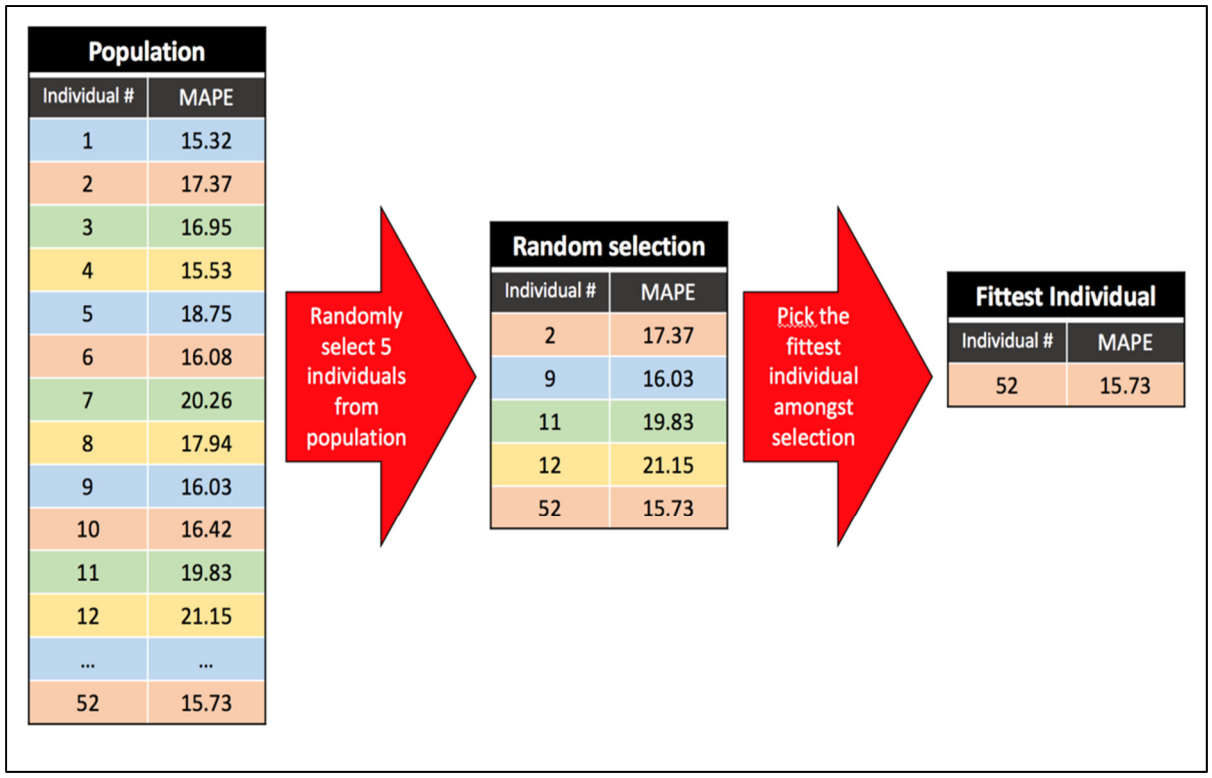

Figure 4.3 Selection Process

4.8.4 Selection

During each successive generation, individuals of the current population are selected to breed a new population (hence the term "generation"). For the purpose of our workload modeling problem, we proceed by tournament selection (Figure 4.3). To do so, we pick one candidate with the fittest solution amongst x randomly selected individuals. The process is then repeated to select a second candidate. Both candidates become parents to generate a new offspring in the crossover operator.

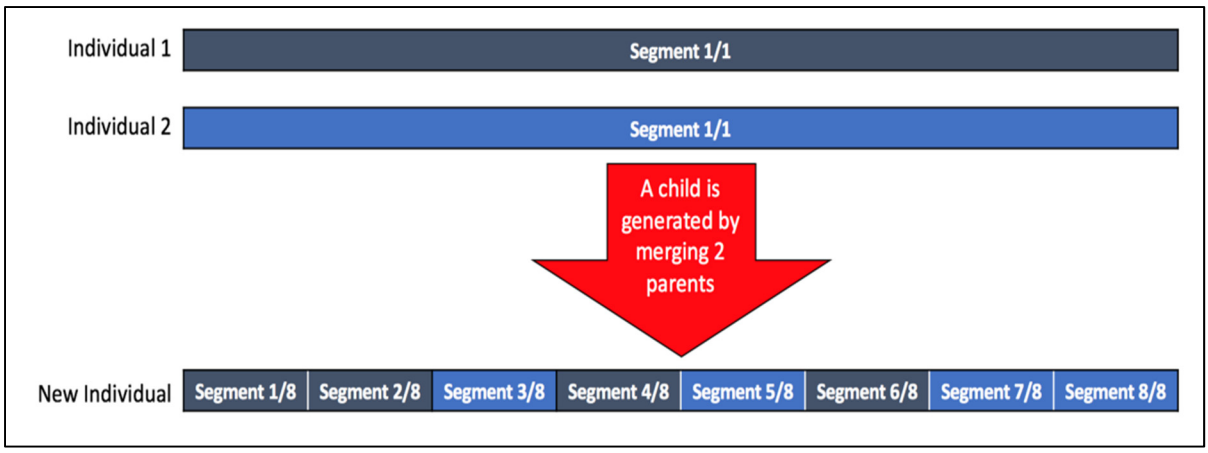

Figure 4.4 Crossover Operation

4.8.5 Crossover Operator

The crossover operator generates a next generation population of solutions from those selected through tournament selection. As shown in Figure 4.4, segments are chosen randomly, based on a uniform rate of ρ_c (probability of selecting segments from one parent over the other), from both parents I_1 and I_2 to generate a new offspring.

4.8.6 Mutation Operator

The mutation operator generates new estimated values from a randomly selected Hull-White model for randomly selected segments of an individual. It is based on a mutation rate of ρ_m (probability of mutating a segment). Figure 4.5 depicts how new workload estimation values of segments 4/8 and 6/8 are modified during the mutation process.

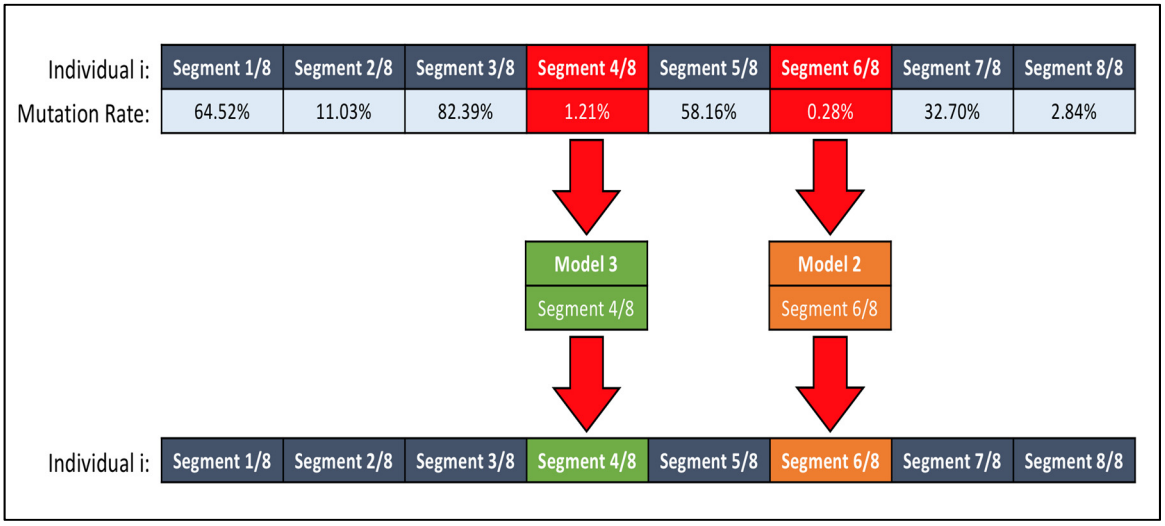

Figure 4.5 Mutation Operator

4.8.7 Algorithm

Algorithm 4.1 describes the proposed process for workload modeling with GA. It starts by generating a random population of candidate workload values (Line 2) then evaluates the fitness function, which is the MAPE for each individual (Line 3 to 9). Afterwards candidates are selected through a tournament process (Lines 13 and 14) to go into the evolution process of crossover (Line 15) and mutation (Line 18). The fitness of these individuals forming the new population is then calculated (Line 20 to 26). The process is repeated until the maximum number of generations (G) is achieved. Finally, the algorithm returns the fittest individual having the least MAPE found throughout the generated populations (Line 29).

Algorithm 4.1 Load Modeling GA

Algorithm 4.1: *LoadModelingGA(L,M,N,D,W,G,T, ρc, ρm)*

Input: L:= Individual length, M := Set of Hull-White models, N := population size, D:= Set of observed data, W := Set of segments, G:= Max generations, T := Tournament, size, ρc:= crossover rate, ρm:= mutation rate
Output: *S*:= fittest individual.

```
1:  Initialize population index k := 0
2:  Generate population Pk := GenerateRandomPop(N, L, M)
3:  for each individual ∈ Pk do
4:    e := Evaluate fitness of individual
5:    if e < fitness of current fittest individual then
6:      fitness of current fittest individual := e
7:      S := i
8:    end if
9:  end for
10:   while k < G do
11:     Insert S into Pk+1
12:     for i = 1 : i ≤ N do
13:       Pick candidate1 := tournamentSelection(Pk , T)
14:       Pick candidate2 := tournamentSelection(Pk , T)
15:       crossover candidate1 and candidate2 to produce new
          offspring and insert offspring into Pk+1
16:     end for
17:     for each individual ∈ Pk+1 do
18:       Mutate segments at random rate ≤ ρm
19:     end for
20:     for each individual ∈ Pk+1 do
21:       e := Evaluate fitness of individual
22:       if e < fitness of current fittest individual then
23:         fitness of current fittest individual := e
24:         S := i
25:       end if
26:     end for
27:     Increment k := k + 1
28:   end while
29:   return S the fittest individual
```

4.9 Kalman-SVR Workload Estimation

Beside Hull-White-GA, we also propose and study a combination of Kalman filter (Hu et al., 2013) and support vector regression (SVR) (Cortes & Vapnik, 1995) for workload estimation. Kalman Filter is a well-known model to estimate a hidden state x of system indirectly from measured data and it can integrate data from as many measurements as available (Hu et al., 2013). The Kalman filter model is defined as follows:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{4.8}$$

with a measurement z,

$$z_k = Hx_k + v_k \tag{4.9}$$

where,

A : Transition matrix from time $k-1$ to k

B : Control matrix

u_{k-1} : Known vector

H : Matrix showing the relationship between z_k and x_k

w_{k-1} and v_k : Process and measurement noise respectively

After filtering observed workloads with the Kalman filter to remove the noise and hence minimize the prediction error, we use SVR to estimate workloads. In SVR, input data are separated into training data classes using linear hyperplanes. If they cannot be linearly separated, then input vectors (observed data) are mapped into high-dimensional feature space using non-linear mapping function (kernel function). SVR identifies the optimal hyperplane that maximizes the margin between the vectors of the considered classes. This optimal hyperplane is defined as linear decision function (find optimal parameters w and b):

$$f(x) = wK(x) + b \tag{4.10}$$

where,

x : Input data, w is the weigth vector and b is the bias parameter

$K(x)$: Kernel function (e.g., linear, Radial Basis Function (RBF))

Algorithm 4.2 EM-KalmanFilter-SVR

Algorithm 2.2: EM-KalmanFilter-SVR*(yi, m, n)*

Input: *yi, m, n*
Output: *filtered estimated data*, *smooth filtered estimated data*

1: Initialize SVR parameters: *Kernel function*, *C*, *Gamma*
2: Initialize data samples for Kalman Filter training and test: *TrainData*, *TestData*
3: Create Kalman filter Object:
kf = **KalmanFilter**(*initial_state_mean*=0, *n_dim_obs*=1)
4: Estimate state with filtering: *estimated_state* = *kf*.**EM**(*TrainData*).**KF**(*TestData*)
5: Estimate state with filtering and smoothing: *smooth_state* =
kf.**EM**(*TrainData*).**SmoothKF**(*TestData*)
6: ***for each*** *m* filtered data
7: *Filter-Prediction*= SVR(*estimated_state* , m, n , *KernelFunction*, *C*, *Gamma*)
8: *SmoothFilter-Prediction*= SVR(smoothed*_state*, *m*, *n* , *KernelFunction*, *C*, *Gamma*)
9: ***end for***
10: ***return*** *filtered estimated data* and *smooth filtered estimated data*

Algorithm 4.2 shows the proposed Expectation Maximization Algorithm (EM) EM-Kalman Filter-SVR. The algorithm starts by initializing SVR parameters (Line 1). The Kernel function model can be Radial Basis Function (RBF), linear, or polynomial. The parameter C trades off misclassification of training examples against simplicity of the decision surface, while Gamma can be defined as the inverse of the radius of influence of samples selected by the model as support vectors (developers scikit-learn, 2013). Then another initialization phase is conducted for the data samples of the Kalman filter. Next, the initial state mean is set to 0 while ndimobs, which is the size of observation space, is set to 1 (Line 3). The estimated state is calculated using EM (Lines 4 and 5). EM, is a meta-algorithm for learning parameters in probabilistic models. It aims to find parameters that maximize the expected likelihood of the observed variables. The EM algorithm is used in this work to estimate model parameters with the Kalman Filter. The algorithm estimates the state with the Kalman filter (Line 4), then estimate

it with filtering and smoothing (Line 5). Afterwards, both the estimated state and the smoothed state are used in SVR for filter prediction, and smooth filter prediction, respectively (Lines 7 and 8). Finally, the algorithm returns the filtered estimated data and the smooth filtered estimated data (Line 10).

4.10 Experimental Results

In this section, we present the results of our experiments.

4.10.1 Use Cases

In our experiments we use CPU workload datasets from two different domains: IT and telecom. We selected these workloads because they differ significantly from each other and we want to evaluate the performance of our approach under different load behaviors, and to assess its general efficiency under various situations. For instance, the IT CPU workloads show sharper variations in short bursts while telecom workloads, on the other hand, show flatter, continuous loads under normal customer demand. This is due to the way that the CPUs handle the different jobs and tasks from each workload domain, as IT CPU loads involve distributed computing while telecom CPU loads generally comprise individual call setup activity. Under critical, unexpected customer demand, however, telecom CPU loads will have dramatic variations and can impact the whole system as customer calls may be dropped. To cover as many scenarios as possible, we use 5 different datasets; 3 from the telecom domain (IMS1, IMS2, IMS3) which have a similar configuration, but with slight variations in the amount of customer calls per second (CPS) generated, and 2 from the IT domain (Google85, ETS Web). The Google85 dataset comes from a single server from a cluster in a Google cloud environment and shows a standard behavior of a CPU workload under normal utilization in a cloud environment. The Web App dataset comes from a virtual machine hosting a web server at Ecole de technologie superieure (ETS) and shows periodicity (cycles) over one week of normal utilization.

4.10.2 Configuration

To demonstrate the proposed hybrid workload modeling approach, we created workload models based on observed CPU workload of virtualized IP Multimedia Subsystem (IMS) and Google clusters. The experiments were performed on a server with a 2.3 GHz Intel Core i7 quad-core processor, 16 GB of RAM, using Matlab R2017a. The following describe the configurations and scenarios used in this paper.

1. Configuration IMS1:
    a. Starts at 150 CPS (calls per second), increment: 50 CPS every 10 sec until 400 CPS;
    b. 400 CPS constant during 100 seconds;
    c. 600 CPS constant during 300 seconds;
    d. 200 CPS decrement: -50 CPS every 50 sec until 50 CPS.

2. Configuration IMS2: IMS1+50 CPS.

3. Configuration IMS3: IMS1-50 CPS.

4. Configuration Google: CPU load of a single machine in a cluster captured and made available by Google.

5. Configuration Web Application: CPU load collected every 30 minutes from a virtual machine hosting a web server at Ecole de technologie superieure (ETS).

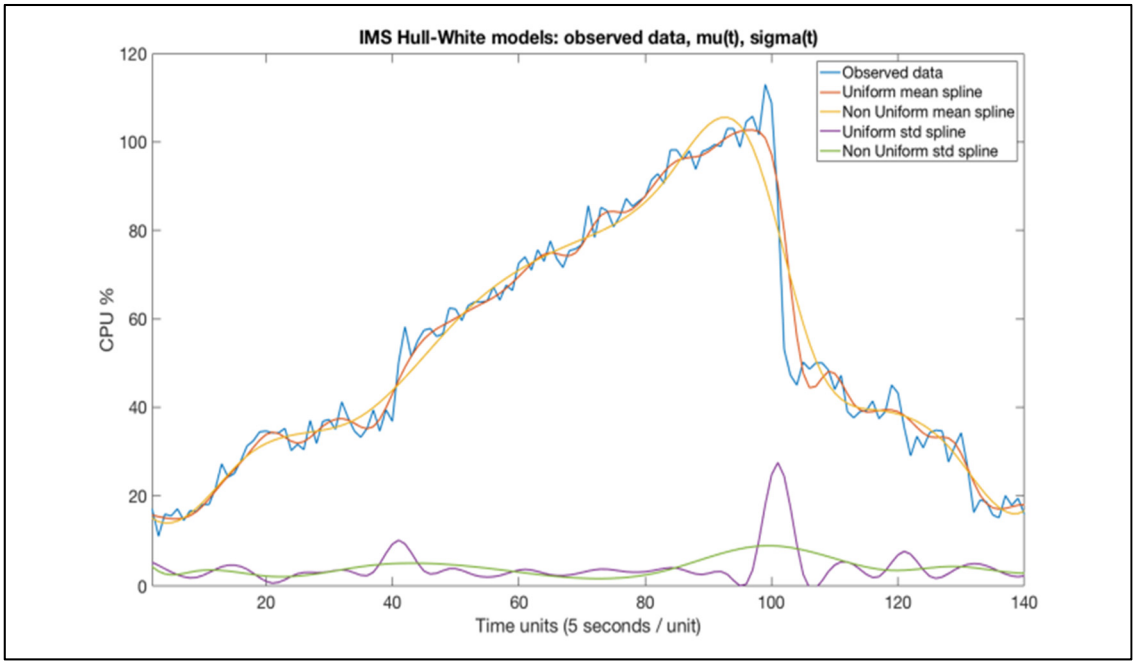

Figure 4.6 IMS Hull-White Models: Observed Data, μ(t) and σ(t)

4.10.3 Results

In the first set of experiments, we use four Hull-White models for workload modeling and then generate estimated workloads accordingly. In the second set of experiments, we combine Hull-White with GA, where the former is used to model workloads and the latter is used to generate optimized workload estimates. Figure 4.6, Figure 4.7, Figure 4.8, Figure 4.9 and Figure 4.10 depict the results for each dataset, while Table 4.1 illustrates the MAPE of the considered models. The results prove that the Hull-White combined with GA is able to generate workloads with the highest fidelity for all tested datasets.

Figure 4.7 IMS: Observed CPU Workloads

Figure 4.8 IMS: Estimated CPU Workload-GA

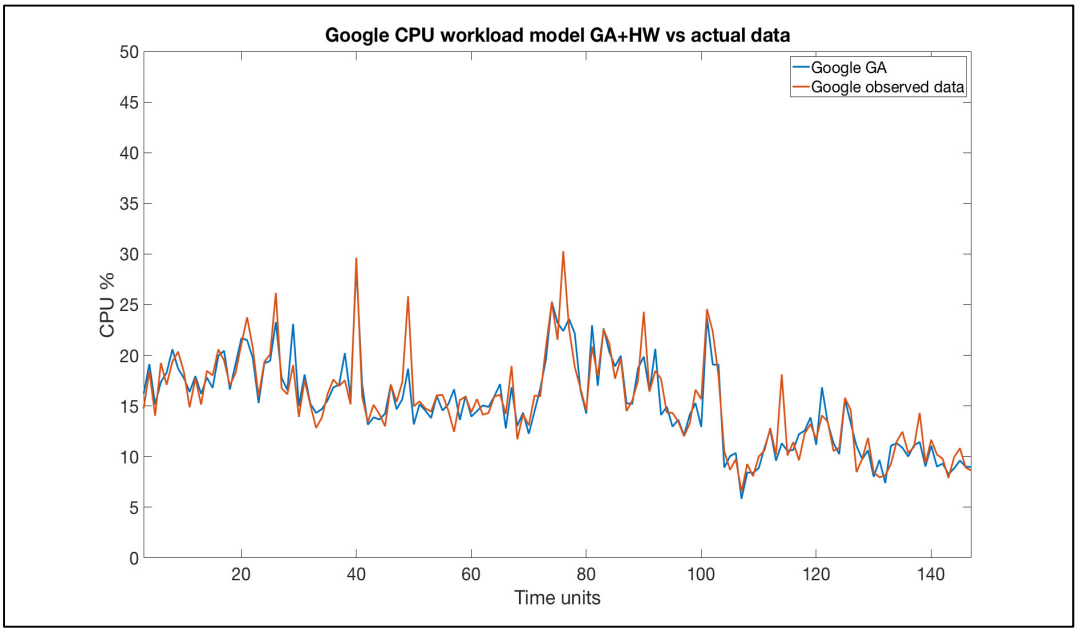

Figure 4.9 Google CPU Workload Model: Hull-White-GA and Observed Data

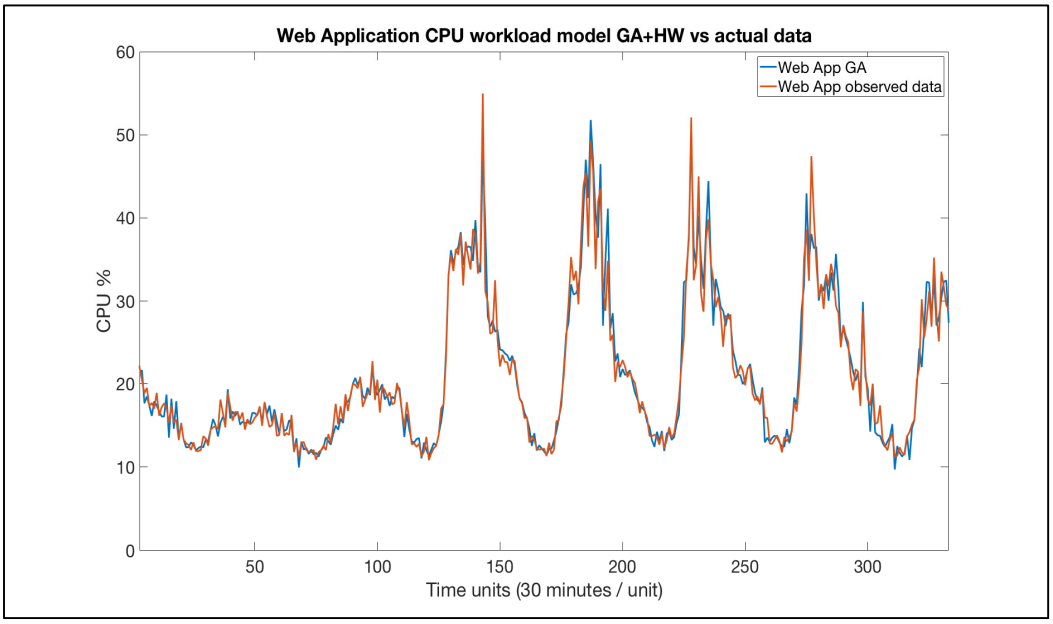

Figure 4.10 Web Application CPU Workload Model: Hull-White-GA and Observed Data

Figure 4.7 and Figure 4.8 show that the estimated workload fits the observed workload, with the exception of some areas where there are sharp CPU increases/decreases. Figure 4.9 shows that the estimated workload follows the same behavior as the observed workload. We notice, however, that the estimated workload is not an exact representation of the observed workload pattern. As from Figure 4.10, in the case of periodic workloads we observe a similar pattern. The estimated workload behavior, however, needs improvement. This is especially noticeable in areas where there are sharp spikes.

Further, we compare all these models with Support Vector Regression (SVR) and Kalman-SVR models. SVR allows data approximation based on statistical learning theory. In this SVR model, the prediction of future resource usage is based on observed data, which we divided into training and prediction sets to generate the estimated workloads. As for Kalman- SVR model, we filter observed data through the Kalman filter, then we predict using SVR. For SVR, we use four observations to train the model and two observations to estimate the next two values. The kernel function is set to RBF since it is more appropriate to nonlinear dataset, while C and Gamma values are fixed to 0.1. These parameters are set through extensive tests performed in order to find the configuration that minimizes the estimation error. As for the Kalman filter, we consider the transition as an identity matrix, we assume a vector of zero control input, and the noise measurement is of the state directly. Therefore, we set $A = 1$, $u = 0$, and $H = 1$ in Equations 4.11 and 4.12, which we define as follows:

$$x_k = x_{k-1} + w_{k-1} \tag{4.11}$$

$$z_k = x_k + v_k \tag{4.12}$$

Figure 4.11, Figure 4.12, Figure 4.13, Figure 4.14 and Figure 4.15 depict the results. Comparing SVR with Kalman-SVR, Table 4.1 proves the efficiency of the latter to generate more accurate results in all the datasets. Yet, the Hull-White-GA proves its efficiency over SVR and Kalman-SVM, showing higher accuracy for IMS and ETS Web datasets. As for the Google CPU workload, Kalman-SVR shows better results in terms of the least mean absolute

percentage error. On the other hand, we compare in Table 4.2 the execution time of all the studied models. The results show that Kalman-SVR outperforms the other models in all the datasets.

To summarize, these experiments show that Hull-White combined with GA is able to provide the highest accuracy for Workload modeling and estimation, but with higher overhead in terms of execution time compared to Kalman-SVR. Yet the latter loses efficiency proportionally to the window size and is more sensitive to large variations and peaks in the observed workload data.

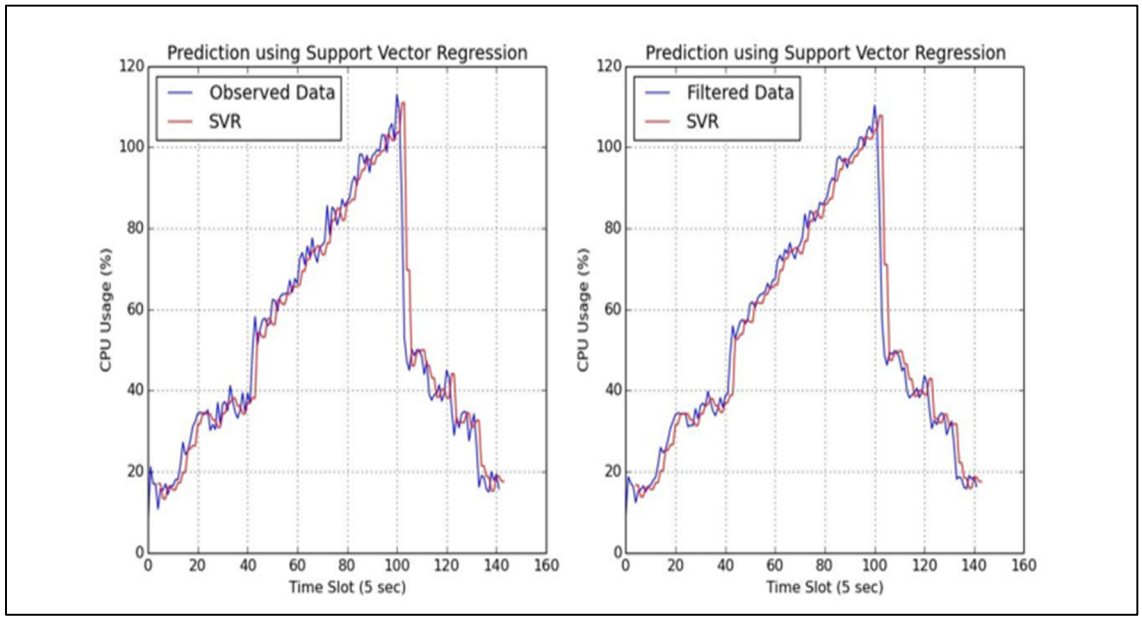

Figure 4.11 IMS1: SVR and Kalman SVR

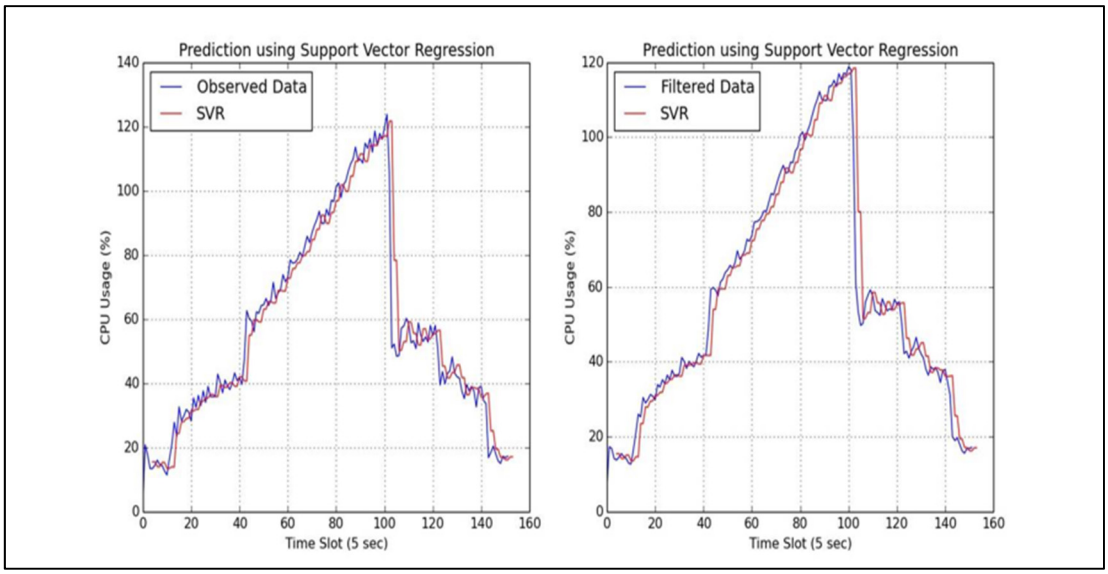

Figure 4.12 IMS2: SVR and Kalman SVR

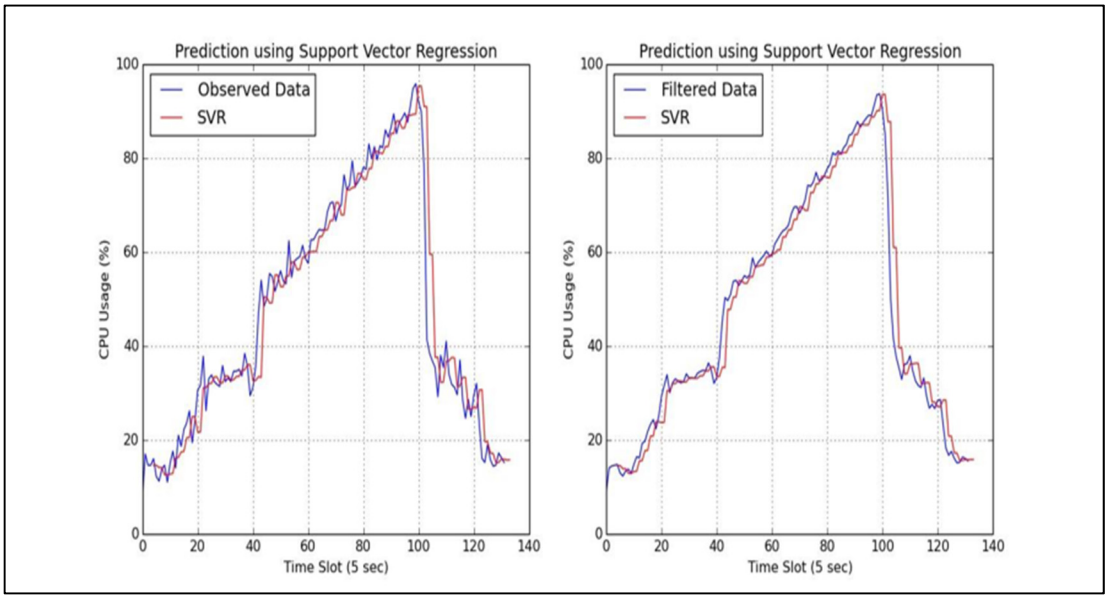

Figure 4.13 IMS3: SVR and Kalman SVR

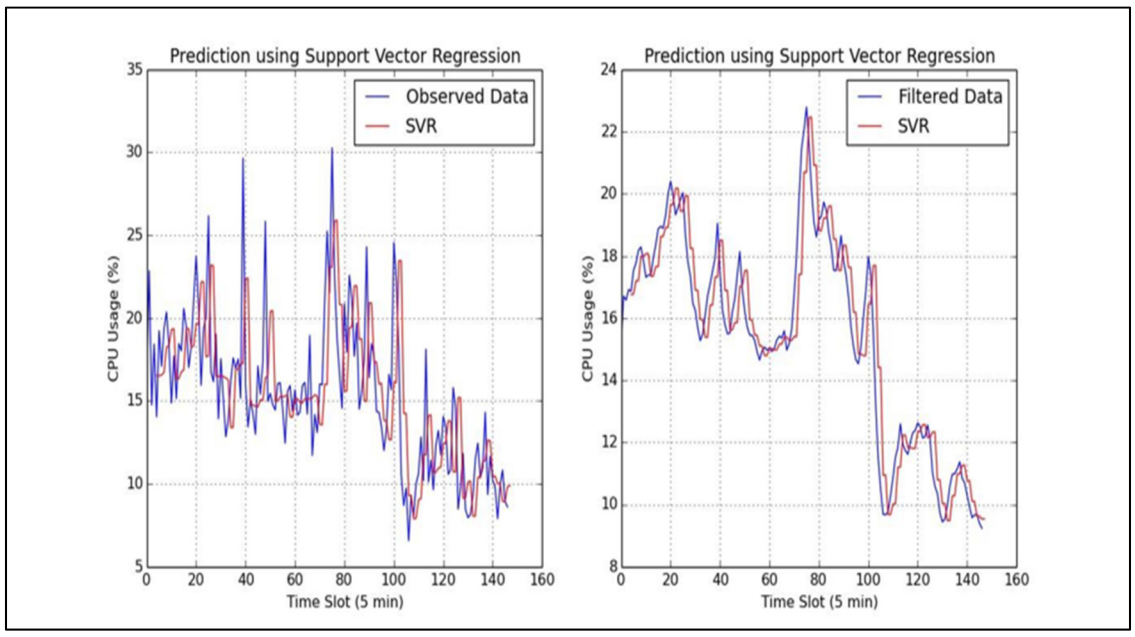

Figure 4.14 Google: SVR and Kalman SVR

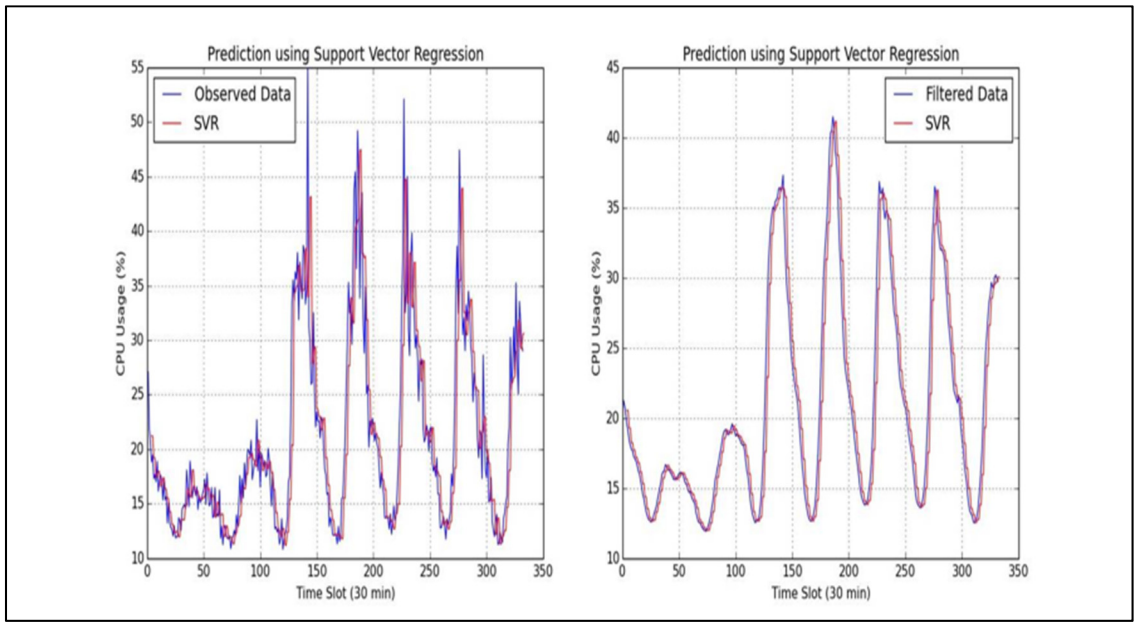

Figure 4.15 Web Application: SVR and Kalman SVR

Table 4.1 Average MAPE of solutions based on 10 simulations

Dataset	GA-Hull White	Hull-White Model 1	Hull-White Model 2	Hull-White Model 3	Hull-White Model 4	SVR	Kalman-SVR
IMS 1	**3,7625**	9,1578	10,8235	9,6495	10,8148	10.27	8.44
IMS 2	**3,931**	10,1615	14,0743	9,4488	14,3537	9.56	7.81
IMS 3	**4,3163**	10,6697	13,6215	10,7017	13,2126	11.15	7.48
Google	7,3226	18,5092	20,2361	17,7301	20,4648	19.19	**5.82**
Web App	**5,3969**	7,6307	9,0903	7,5744	9,1559	11.83	6.70

Table 4.2 Average execution time (s) based on 10 minutes simulations, 60 generations for GA

Dataset	GA-Hull White	SVR	Kalman-SVR
IMS 1	**2,77**	$9{,}60*10^{-3}$	$9{,}16*10^{-3}$
IMS 2	**2,80**	$10{,}11*10^{-3}$	$9{,}46*10^{-3}$
IMS 3	**2,63**	$9{,}19*10^{-3}$	$8{,}41*10^{-3}$
Google	**2,88**	$9{,}81*10^{-3}$	$8{,}99*10^{-3}$
Web App	**4,16**	$22{,}04*10^{-3}$	$21{,}46*10^{-3}$

4.11 Conclusion and Future Work

For dynamic on-demand adjustment and provisioning of resource needs in the Cloud environment, an accurate prediction of the system behavior is needed. The assessment of system behavior requires large amounts of workload data. To address the need for real workload data, something that is hard to obtain, we proposed in this article two novel paradigms for workload emulation, namely a Hull-White model combined with custom genetic algorithm and support vector regression model optimized with Kalman filter. We evaluated both techniques on different datasets of IMS, Google and Web Application CPU workloads. The results have proved the advantage of Hull-White GA model over SVR and Kalman-SVM, showing higher accuracy for IMS and Web App datasets. As for the Google CPU workload

data, Kalman-SVR showed better results in terms of the least mean absolute percentage error. However, for all datasets, Hull-White GA has outperformed both standard SVR and Kalman-SVR with negligible execution time of 0.00181 seconds. Such promising results open the door for a valuable track to examine the proposed hybrid workload modeling approaches on other workload attributes such as RAM and network traffic.

Acknowledgement

This work has been supported in part by Natural Science and Engineering Research Council of Canada (NSERC), in part by Ericsson Canada and in part by Rogers Communication Canada.

CONCLUSION

Dans un environnement infonuagique, des ressources partageables sont mises à la disposition des utilisateurs d'une manière flexible (échelonnement automatique) et selon une stratégie payez à l'usage. La disponibilité des ressources et la charge de trafic imposée par chaque application qui les consomme varient continuellement. Une telle variation requière une gestion proactive des ressources pour garantir la QdS en dépit du caractère dynamique de l'environnement infonuagique qui les héberge. Pour garantir cette QdS, les fournisseurs d'infrastructures infonuagique optent généralement pour une solution de sur-approvisionnement, induisant ainsi une sous-utilisation des ressources pendant les périodes de charge habituelle de trafic et les coûts importants de gestion et d'opération de ces infrastructures, et de consommation d'énergie. Une gestion proactive de ces ressources serait donc cruciale pour assurer la QdS demandée, optimiser l'utilisation des ressources et minimiser les coûts.

Dans cette perspective, cette thèse avait pour objectif principal de définir et valider une approche proactive et adaptive de prédiction de la consommation de ressources tout en assurant les exigences du SLA et en optimisant l'utilisation de ces ressources. En se basant sur des algorithmes d'auto-apprentissage, notre approche s'est révélée suffisamment générique pour être appliquée à des jeux de données variés. Elle comprend principalement: une prédiction automatique de la consommation de ressources, et une détection automatique des comportements inhabituels dans l'utilisation de ces ressources.

Dans le premier article présenté au chapitre 2, nous avons proposé une approche automatique, proactive et adaptive pour la prédiction de la consommation en ressources en utilisant l'apprentissage automatique. Sans aucune connaissance préalable du système/application et de son environnement où il/elle est déployée/e, notre approche est suffisamment générique pour être applicable à différents systèmes virtualisés (jeux de données). Elle permet aussi une lecture automatique des données collectées grâce au principe de la fenêtre coulissante dont la taille est

déterminée automatiquement par l'algorithme génétique. Notre approche comporte également un ajustement automatique de la prédiction qui se base sur le calcul de la probabilité de l'erreur et un ajustement automatique supplémentaire (*padding*) afin de réduire les sous/sur-approvisionnements.

Dans le deuxième article décrit dans le chapitre 3, nous avons proposé une approche pour détecter et évaluer automatiquement les comportements anormaux, les profils (*Patterns*) et la périodicité de ces changements. Dans cette approche, une mise en correspondance (*Mapping*) des métriques de niveau ressources avec celles de niveau services est réalisée automatiquement afin de générer des notifications spécifiant le type de décision et sa priorité dans le but d'aider dans la planification des ressources.

Dans le troisième article présenté dans le chapitre 4, nous avons développé une approche permettant d'estimer la charge de trafic laquelle combine un filtre de Kalman avec une approche de régression à vecteurs de support (SVR-Support Vector regression). L'analyse présentée dans ce chapitre a permis de développer une nouvelle approche automatique de détection de périodicité dans les charges de trafic ayant différentes amplitudes, formes et périodicité qui a été utilisée dans l'approche de détection des comportements anormaux décrites dans le chapitre 3.

Les contributions de cette thèse se résument comme suit:

Contributions techniques:

- un nouvel algorithme pour une prédiction automatique et adaptive de la demande en ressources sans aucune connaissance ou hypothèse préalable sur le système ou son comportements interne.
- un ajustement automatique et adaptif de la prédiction ainsi que la détermination automatique des tailles optimales de la fenêtre coulissante et des données prédites.
- un nouvel algorithme pour une détection automatiques des variations anormales et une mise en correspondance des métriques du niveau de services avec celles du niveau des ressources ainsi qu'une détection de la périodicité.

- une évaluation efficace et automatique de l'état du système et la génération de notifications induisant soit une adaptation des ressources ou des alertes signalant un état inhabituel nécessitant un suivi plus poussé de l'état de l'infrastructure.

Publications et brevet:

1. Souhila Benmakrelouf, Nadjia Kara, Hanine Tout, Rafi Rabipour and Claes Edstrom. (2019). Resource needs prediction in virtualized systems: generic proactive and self-adaptation solution. Elsevier Journal of Network and Computer Applications (IF-5.273), 148, 1-16.

2. Benmakrelouf, **S.**, St-Onge, C., Kara, N., Tout, H., Edstrom, C., & Lemieux, Y. (2020). Abnormal behavior detection using resource level to service level metrics mapping in virtualized systems. Future Generation Computer Systems (IF-5.768), 102, 680-700.

3. Cédric St-Onge, Souhila Benmakrelouf**,** Nadjia Kara, Hanine Tout, Claes Edstrom and Rafi Rabipour. (2020) Advanced workload modelling for cloud systems. Accepted for publication in Journal of Cloud Computing (IF-2.960).

4. Souhila Benmakrelouf and Nadjia Kara. Resource Needs Prediction in Virtualized Systems: Generic Proactive and Self-Adaptive Solution. Regular Patent reference number P73928 WIPO, January 2019.

Chacune des contributions de cette thèse ouvre de nouvelles perspectives de recherche. Dans ce qui suit, nous listons quelques exemples de ces perspectives :

1. Basés sur l'apprentissage automatique, la prédiction dynamique et adaptive que nous avons proposée ainsi que la détermination des fenêtres de données d'apprentissage ont permis de fournir une estimation de la demande future des ressources avec une bonne précision. Toutefois, l'investigation de nouvelles méthodes basées sur l'apprentissage automatique et combinant plusieurs métriques au niveau service (ex., charge de trafic, latence, perte de paquets) et au niveau ressources (ex., CPU, mémoire, bande passante) pourrait améliorer la

précision de la prédiction, et inclure des systèmes nécessitant la consommation intensive de plusieurs ressources à la fois (ex., CPU et bande passante). Dans ce cas, il est nécessaire de trouver un compromis entre la complexité des approches proposées et l'efficacité de la prédiction de la consommation de ressources.

2. L'ajustement de la prédiction réalisé dans cette thèse a permis de minimiser les situations de sous-estimation. Mais trouver un compromis entre la réduction de la sous-estimation et de la surestimation ainsi que la diminution du délai d'ajustement nécessitent de nouvelles investigations.
3. L'algorithme proposé d'analyse et de mise en correspondance a permis de détecter les variations significatives et inhabituelles dans le comportement de systèmes (jeux de données). Néanmoins, le développement de nouvelles techniques permettant de détecter des valeurs aberrantes, et inclure de nouvelles métriques collectés à partir des outils de monitorage afin de corroborer les notifications issues de différentes ressources et améliorer ainsi leur précision. Également, une nouvelle approche pour un ajustement automatique de la taille de la fenêtre coulissante pourrait faire l'objet d'une future étude.
4. En tenant compte des résultats de la prédiction, de la détection des variations et de la périodicité des charges de trafic, une planification automatique des ressources permettra une fois réalisée d'estimer la quantité de ressources et l'intervalle de temps appropriés pour déployer ces ressources.

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUE

Abramowitz, M., & Stegun, I. A. (1972). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*.
Repéré à http://people.math.sfu.ca/~cbm/aands/abramowitz_and_stegun.pdf

Aggarwal, C. C. (2013). *Outlier Analysis*. New York: Springer-Verlag doi: 10.1007/978-1-4614-6396-2

Aggarwal, C. C. (2015). Outlier Analysis. Dans *Data Mining* (pp. 237-263). doi: 10.1007/978-3-319-14142-8_8

Ahdesmaki, M., Lahdesmaki, H., Pearson, R., Huttunen, H., & Yli-Harja, O. (2005). Robust detection of periodic time series measured from biological systems. *BMC Bioinformatics, 6*, 117. doi: 10.1186/1471-2105-6-117.
Repéré à https://www.ncbi.nlm.nih.gov/pubmed/15892890

Ahmed, M., Mahmood, A. N., & Islam, M. R. (2016). A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems, 55*, 278-288. doi: 10.1016/j.future.2015.01.001

Ahmed, M., Naser Mahmood, A., & Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications, 60*, 19-31. doi: 10.1016/j.jnca.2015.11.016

Aldribi, A., Traoré, I., Moa, B., & Nwamuo, O. (2020). Hypervisor-based cloud intrusion detection through online multivariate statistical change tracking. *Computers & Security, 88*. doi: 10.1016/j.cose.2019.101646

Amiri, M., & Mohammad-Khanli, L. (2017). Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications, 82*, 93-113. doi: 10.1016/j.jnca.2017.01.016

An, C., Zhou, J., Liu, S., & Geihs, K. (2016). A multi-tenant hierarchical modeling for cloud computing workload. *Intelligent Automation & Soft Computing, 22*(4), 579-586. doi: 10.1080/10798587.2016.1152774

Areeg, S., & Pahl, C. (2019, May 05, 2019 to May 09, 2019). *Anomaly Detection and Analysis for Clustered Cloud Computing Reliability* présentée à CLOUD COMPUTING 2019 : The Tenth International Conference on Cloud Computing, GRIDs, and Virtualization, Venice, Italy.

Bahga, A., & Madisetti, V. K. (2011). Synthetic Workload Generation for Cloud Computing Applications. *Journal of Software Engineering and Applications, 04*(07), 396-410. doi: 10.4236/jsea.2011.47046

Benmakrelouf, S., & Kara, N. (2019). *Brevet*.

Benmakrelouf, S., Kara, N., Tout, H., Rabipour, R., & Edstrom, C. (2019). Resource needs prediction in virtualized systems: Generic proactive and self-adaptive solution. *Journal of Network and Computer Applications, 148*. doi: 10.1016/j.jnca.2019.102443

Benmakrelouf, S., St-Onge, C., Kara, N., Tout, H., Edstrom, C., & Lemieux, Y. (2020). Abnormal behavior detection using resource level to service level metrics mapping in virtualized systems. *Future Generation Computer Systems, 102*, 680-700. doi: 10.1016/j.future.2019.07.051

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., & Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications, 9*(1). doi: 10.1186/s13174-018-0087-2

Braverman, M. (2011). COS597D: Information Theory in Computer Science - Lecture 3. Repéré à https://www.cs.princeton.edu/courses/archive/fall11/cos597D/L03.pdf

Cao, L. J., & Tay, F. H. (2003). Support vector machine with adaptive parameters in financial time series forecasting. *IEEE Trans Neural Netw, 14*(6), 1506-1518. doi: 10.1109/TNN.2003.820556.
Repéré à https://www.ncbi.nlm.nih.gov/pubmed/18244595

Chanda, A. K., Ahmed, C. F., Samiullah, M., & Leung, C. K. (2017). A new framework for mining weighted periodic patterns in time series databases. *Expert Systems with Applications, 79*, 207-224. doi: 10.1016/j.eswa.2017.02.028

Chanda, A. K., Saha, S., Nishi, M. A., Samiullah, M., & Ahmed, C. F. (2015). An efficient approach to mine flexible periodic patterns in time series databases. *Engineering Applications of Artificial Intelligence, 44*, 46-63. doi: 10.1016/j.engappai.2015.04.014

Chen, J., & Wang, Y. (2020). An Adaptive Short-Term Prediction Algorithm for Resource Demands in Cloud Computing. *IEEE Access, 8*, 53915-53930. doi: 10.1109/access.2020.2981011

Chen, X., Zhu, F., Chen, Z., Min, G., Zheng, X., & Rong, C. (2020). Resource Allocation for Cloud-Based Software Services Using Prediction-Enabled Feedback Control with Reinforcement Learning. *IEEE Transactions on Cloud Computing*, 1-1. doi: 10.1109/tcc.2020.2992537

Chien, W.-C., Lai, C.-F., & Chao, H.-C. (2019). Dynamic Resource Prediction and Allocation in C-RAN With Edge Artificial Intelligence. *IEEE Transactions on Industrial Informatics, 15*(7), 4306-4314. doi: 10.1109/tii.2019.2913169

Containers, L. (2017). Infrastructure for container projects. Repéré le 2017-06-06 à https://linuxcontainers.org/

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*(3), 273-297. doi: 10.1007/bf00994018

da Rosa Righi, R., Correa, E., Gomes, M. M., & da Costa, C. A. (2020). Enhancing performance of IoT applications with load prediction and cloud elasticity. *Future Generation Computer Systems, 109*, 689-701. doi: 10.1016/j.future.2018.06.026

Dias, Z., & Dias, U. (2015). Sorting by Prefix Reversals and Prefix Transpositions. *Discrete Applied Mathematics, 181*, 78-89. doi: 10.1016/j.dam.2014.09.004

Dunn, P. F. (2014). *Measurement and Data Analysis for Engineering and Science*. CRC Press, Taylor & Francis. doi: 10.1201/b22182

Dutta, A. K., Hasan, M., & Rahman, M. S. (2013). Prefix transpositions on binary and ternary strings. *Information Processing Letters, 113*(8), 265-270. doi: 10.1016/j.ipl.2013.01.017

Elfeky, M. G., Aref, W. G., & Elmagarmid, A. K. (2005). Periodicity detection in time series databases. *IEEE Transactions on Knowledge and Data Engineering, 17*(7), 875-887. doi: 10.1109/tkde.2005.114

Emeakaroha, V. C., Brandic, I., Maurer, M., & Dustdar, S. (2010). *Low level Metrics to High level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments* présentée à 2010 International Conference on High Performance Computing & Simulation. doi: 10.1109/hpcs.2010.5547150

ETSI. (2013). *Network Functions Virtualisation (NFV);Architectural Framework*. Repéré à https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

ETSI GS, N.-M. (2014). *Network Functions Virtualization (NFV); Management and Orchestration.* Repéré à https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

Feitelson, D. G. (2015). *Workload modeling for computer systems performance evaluation*. Cambridge University Press.

Fokus, F. (2014). Open Source IMS Core by cnd. Repéré le 2017-06-06 à http://www.openimscore.org/

Fu, T.-c. (2011). A review on time series data mining. *Engineering Applications of Artificial Intelligence, 24*(1), 164-181. doi: 10.1016/j.engappai.2010.09.007

Gambi, A. (2012). *Kriging-based self-adaptive controllers for the cloud* (Università della Svizzera italiana). Repéré à https://doc.rero.ch/record/32769

Gambi, A., Pezze, M., & Toffetti, G. (2016). Kriging-Based Self-Adaptive Cloud Controllers. *IEEE Transactions on Services Computing, 9*(3), 368-381. doi: 10.1109/tsc.2015.2389236

Gandhi, A., Yuan, C., Gmach, D., Arlitt, M., & Marwah, M. (2011). *Minimizing data center SLA violations and power consumption via hybrid resource provisioning* présentée à 2011 International Green Computing Conference and Workshops. doi: 10.1109/igcc.2011.6008611

Gao, J., Wang, H., & Shen, H. (2020). Task Failure Prediction in Cloud Data Centers Using Deep Learning. *IEEE Transactions on Services Computing*, 1-1. doi: 10.1109/tsc.2020.2993728

Gayraud, R., & Jacques, O. (2014). SIPp. Repéré le 2017-06-06 à http://sipp.sourceforge.net/

Gil Herrera, J., & Botero, J. F. (2016). Resource Allocation in NFV: A Comprehensive Survey. *IEEE Transactions on Network and Service Management, 13*(3), 518-532. doi: 10.1109/tnsm.2016.2598420

Goudarzi, H., & Pedram, M. (2016). Hierarchical SLA-Driven Resource Management for Peak Power-Aware and Energy-Efficient Operation of a Cloud Datacenter. *IEEE Transactions on Cloud Computing, 4*(2), 222-236. doi: 10.1109/tcc.2015.2474369

Gratton, Y. (2002). Le krigeage: la méthode optimale d'interpolation spatiale. *Les articles de l'Institut d'Analyse Géographique, 1*(4), 1-4.

Hawkins, D. M. (1980). *Identification of Outliers*. Springer, Dordrecht. doi: 10.1007/978-94-015-3994-4

Hoong, P. K., Tan, I. K., & Keong, C. Y. (2012). Bittorrent Network Traffic Forecasting With ARMA. *International Journal of Computer Networks & Communications, 4*(4), 143-156.

Hsieh, S.-Y., Liu, C.-S., Buyya, R., & Zomaya, A. Y. (2020). Utilization-prediction-aware virtual machine consolidation approach for energy-efficient cloud data centers. *Journal of Parallel and Distributed Computing, 139*, 99-109. doi: 10.1016/j.jpdc.2019.12.014

Hu, R., Jiang, J., Liu, G., & Wang, L. (2013). *CPU Load Prediction Using Support Vector Regression and Kalman Smoother for Cloud* présentée à 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops. doi: 10.1109/icdcsw.2013.60

Huang, C.-J., Guan, C.-T., Chen, H.-M., Wang, Y.-W., Chang, S.-C., Li, C.-Y., & Weng, C.-H. (2013). An adaptive resource management scheme in cloud computing. *Engineering Applications of Artificial Intelligence, 26*(1), 382-389. doi: 10.1016/j.engappai.2012.10.004

Iglewicz, B., & Hoaglin, D. C. *How to detect and handle outliers*. ASQC Quality Press.

Intel Developer, Z. Quality Metrics for Multi-class Classification Algorithms. Repéré le 2017-11-13 à https://software.intel.com/en-us/daal-programming-guide-quality-metrics-for-multi-class-classification-algorithms

Iqbal, M. F., & John, L. K. (2012). *Power and performance analysis of network traffic prediction techniques* présentée à 2012 IEEE International Symposium on Performance Analysis of Systems & Software. doi: 10.1109/ispass.2012.6189212

Islam, S., Keung, J., Lee, K., & Liu, A. (2012). Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems, 28*(1), 155-162. doi: 10.1016/j.future.2011.05.027

Jing, J., Jie, L., Guangquan, Z., & Guodong, L. (2013). *Optimal Cloud Resource Auto-Scaling for Web Applications* présentée à 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. doi: 10.1109/CCGrid.2013.73

Kaur, G., Bala, A., & Chana, I. (2019). An intelligent regressive ensemble approach for predicting resource usage in cloud computing. *Journal of Parallel and Distributed Computing, 123*, 1-12. doi: 10.1016/j.jpdc.2018.08.008

Khaledur Rahman, M., & Sohel Rahman, M. (2015). Prefix and suffix transreversals on binary and ternary strings. *Journal of Discrete Algorithms, 33*, 160-170. doi: 10.1016/j.jda.2015.03.009

Krige, D. G. (1951). A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy, 52*(6), 119 – 139.

León-Castro, E., Avilés-Ochoa, E., & Merigó, J. M. (2018). Induced Heavy Moving Averages. *International Journal of Intelligent Systems, 33*(9), 1823-1839. doi: 10.1002/int.21916

Li, G., & Wang, Y. (2012). *Differential Kullback-Leibler Divergence Based Anomaly Detection Scheme in Sensor Networks* présentée à 2012 IEEE 12th International Conference on Computer and Information Technology. doi: 10.1109/cit.2012.197

Liang, J., Cao, J., Wang, J., & Xu, Y. (2011). *Long-Term CPU Load Prediction* présentée à 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing. doi: 10.1109/dasc.2011.28

Liu, G., Xu, B., & Chen, H. (2014). An indicator kriging method for distributed estimation in wireless sensor networks. *International Journal of Communication Systems, 27*(1), 68-80. doi: 10.1002/dac.2344

Liu, J., Shen, H., & Chen, L. (2016). *CORP: Cooperative Opportunistic Resource Provisioning for Short-Lived Jobs in Cloud Systems* présentée à 2016 IEEE International Conference on Cluster Computing (CLUSTER). doi: 10.1109/cluster.2016.65

Lloyd, W., Pallickara, S., David, O., Lyon, J., Arabi, M., & Rojas, K. (2013). Performance implications of multi-tier application deployments on Infrastructure-as-a-Service clouds: Towards performance modeling. *Future Generation Computer Systems, 29*(5), 1254-1264. doi: 10.1016/j.future.2012.12.007

Ma, K., Li, X., Srinivasa, S. R., Liu, Y., Sampson, J., Xie, Y., & Narayanan, V. (2017). *Spendthrift: Machine learning based resource and frequency scaling for ambient energy harvesting nonvolatile processors* présentée à 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). doi: 10.1109/aspdac.2017.7858402

Magalhães, D., Calheiros, R. N., Buyya, R., & Gomes, D. G. (2015). Workload modeling for resource usage analysis and simulation in cloud computing. *Computers & Electrical Engineering, 47*, 69-81. doi: 10.1016/j.compeleceng.2015.08.016

Matheron, G. (1963). Principles of geostatistics. *Economic Geology, 58*(8), 1246-1266. doi: 10.2113/gsecongeo.58.8.1246

MathWorks. (2017a). Numerical integration. Repéré le 2017-04-28 à https://www.mathworks.com/help/matlab/ref/integral.html#btdd9x5

MathWorks. (2017b). Numerically evaluate integral, adaptive Simpson quadrature. Repéré le 2017-04-28 à https://www.mathworks.com/help/matlab/ref/quad.html

Menasce, D. A., Almeida, V. A., Dowdy, L. W., & Dowdy, L. (2004). *Performance by Design: Computer Capacity Planning By Example* Prentice Hall Professional.

Mitsuo, G., & Runwei, C. (2000). *Genetic algorithms and engineering optimization* (Wiley and Sons. éd.). New York, N.Y.

Mkwawa, I. M., & Kouvatsos, D. D. (2008). *Performance Modelling and Evaluation of Handover Mechanism in IP Multimedia Subsystems* présentée à 2008 Third International Conference on Systems and Networks Communications. doi: 10.1109/icsnc.2008.48

Moreno, I. S., Garraghan, P., Townend, P., & Jie, X. (2013). *An Approach for Characterizing Workloads in Google Cloud to Derive Realistic Resource Utilization Models* présentée à 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering. doi: 10.1109/sose.2013.24

Nguyen, H., Shen, Z., Gu, X., Subbiah, S., & Wilkes, J. (2013). AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service. Dans *10th International Conference on Autonomic Computing* (pp. 69-82). USENIX.

Nishi, M. A., Ahmed, C. F., Samiullah, M., & Jeong, B.-S. (2013). Effective periodic pattern mining in time series databases. *Expert Systems with Applications, 40*(8), 3015-3027. doi: 10.1016/j.eswa.2012.12.017

Qazi, K., Li, Y., & Sohn, A. (2014). *Workload Prediction of Virtual Machines for Harnessing Data Center Resources* présentée à 2014 IEEE 7th International Conference on Cloud Computing. doi: 10.1109/cloud.2014.76

Rasheed, F., & Alhajj, R. (2008). STNR: A suffix tree based noise resilient algorithm for periodicity detection in time series databases. *Applied Intelligence, 32*(3), 267-278. doi: 10.1007/s10489-008-0144-9

Rasheed, F., & Alhajj, R. (2014). A framework for periodic outlier pattern detection in time-series sequences. *IEEE Trans Cybern, 44*(5), 569-582. doi: 10.1109/TSMCC.2013.2261984.
Repéré à https://www.ncbi.nlm.nih.gov/pubmed/23757597

Reiss, C., Wilkes, J., & Hellerstein, J. L. (2014). *Google cluster-usage traces: format+ schema*.
Repéré à https://drive.google.com/file/d/0B5g07T_gRDg9Z0lsSTEtTWtpOW8/view

Salem, O., Nait-Abdesselam, F., & Mehaoua, A. (2012). *Anomaly detection in network traffic using Jensen-Shannon divergence* présentée à 2012 IEEE International Conference on Communications (ICC). doi: 10.1109/icc.2012.6364602

scikit-learn, d. RBF SVM parameters, . Repéré le 2017-10-16 à https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Scikit-learn, d. (2013). Support Vector Machines. Repéré le 2019-03-15 à https://scikit-learn.org/stable/modules/svm.html#svm-regression

Shaw, S. B., & Singh, A. K. (2015). Use of proactive and reactive hotspot detection technique to reduce the number of virtual machine migration and energy consumption in cloud data center. *Computers & Electrical Engineering, 47*, 241-254. doi: 10.1016/j.compeleceng.2015.07.020

Shen, Z., Subbiah, S., Gu, X., & Wilkes, J. (2011). *CloudScale: elastic resource scaling for multi-tenant cloud systems* présentée à Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC '11. doi: 10.1145/2038916.2038921

Shyam, G. K., & Manvi, S. S. (2016). Virtual resource prediction in cloud environment: A Bayesian approach. *Journal of Network and Computer Applications, 65*, 144-154. doi: 10.1016/j.jnca.2016.03.002

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing, 14*(3), 199-222. doi: 10.1023/b:Stco.0000035301.49549.88

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management, 45*(4), 427-437. doi: 10.1016/j.ipm.2009.03.002

Srinivasan, B. V., Duraiswami, R., & Murtugudde, R. (2008). Efficient kriging for real-time spatio-temporal interpolation. Dans *Proceedings of the 20th Conference on Probability and Statistics in the Atmospheric Sciences* (pp. 228-235). American Meteorological Society.

St-Onge, C., Benmakrelouf, S., Kara, N., Tout, H., Edstrom, C., & Rabipour, R. (2020). *Advanced workload modelling for cloud systems*. Journal of Cloud Computing.

St-Onge, C., Kara, N., Abdel Wahab, O., Edstrom, C., & Lemieux, Y. (2020). Detection of time series patterns and periodicity of cloud computing workloads. *Future Generation Computer Systems, 109*, 249-261. doi: 10.1016/j.future.2020.03.059

Stavroulas, Y., Charilas, D., Xydas, G., Varvarigou, T., Psychas, A., Kousiouris, G., . . . Bouras, I. (2019). *Mapping of Quality of Service Requirements to Resource Demands for IaaS* présentée à Proceedings of the 9th International Conference on Cloud Computing and Services Science. doi: 10.5220/0007676902630270

Tahmasbi, R., & Hashemi, S. M. (2014). Modeling and Forecasting the Urban Volume Using Stochastic Differential Equations. *IEEE Transactions on Intelligent Transportation Systems, 15*(1), 250-259. doi: 10.1109/tits.2013.2278614

Takeuchi, J., & Yamanishi, K. (2006). A unifying framework for detecting outliers and change points from time series. *IEEE Transactions on Knowledge and Data Engineering, 18*(4), 482-492. doi: 10.1109/tkde.2006.1599387

Toumi, H., Brahmi, Z., & Gammoudi, M. M. (2020). *Extended Hoeffding Adaptive Tree based-Server Load Prediction in Cloud Computing environment* présentée à Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region. doi: 10.1145/3368474.3368475

Tout, H., Talhi, C., Kara, N., & Mourad, A. (2019). Selective Mobile Cloud Offloading to Augment Multi-Persona Performance and Viability. *IEEE Transactions on Cloud Computing, 7*(2), 314-328. doi: 10.1109/tcc.2016.2535223

Tran, D., Tran, N., Nguyen, B. M., & Le, H. (2016). *PD-GABP — A novel prediction model applying for elastic applications in distributed environment* présentée à 2016 3rd National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS). doi: 10.1109/nics.2016.7725658

Van Beers, W. C. M., & Kleijnen, J. P. C. (2004). *Kriging Interpolation in Simulation: A Survey* présentée à Proceedings of the 2004 Winter Simulation Conference, 2004. doi: 10.1109/wsc.2004.1371308

Vashistha, A., & Verma, P. (2020). *A Literature Review and Taxonomy on Workload Prediction in Cloud Data Center* présentée à 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence). doi: 10.1109/Confluence47617.2020.9057938

Wang, T., Wei, J., Zhang, W., Zhong, H., & Huang, T. (2014). Workload-aware anomaly detection for Web applications. *Journal of Systems and Software, 89*, 19-32. doi: 10.1016/j.jss.2013.03.060

Wang, W., Huang, X., Qin, X., Zhang, W., Wei, J., & Zhong, H. (2012). *Application-Level CPU Consumption Estimation: Towards Performance Isolation of Multi-tenancy Web Applications* présentée à 2012 IEEE Fifth International Conference on Cloud Computing. doi: 10.1109/cloud.2012.81

WebFinance, B. D. Moving average – Definition. Repéré le 2018-08-15 à : http://www.businessdictionary.com/definition/moving-average.html

Wei, W. S. (1990). *Time series analysis. Univariate and multivariate methods* (Addison. Wesley publishingcomp. éd.).

Wei, Z., Tao, T., ZhuoShu, D., & Zio, E. (2013). A dynamic particle filter-support vector regression method for reliability prediction. *Reliability Engineering & System Safety, 119*, 109-116. doi: 10.1016/j.ress.2013.05.021

Yadav, R. M. (2019). Effective analysis of malware detection in cloud computing. *Computers & Security, 83*, 14-21. doi: 10.1016/j.cose.2018.12.005

Yang, H., Luan, Z., Li, W., & Qian, D. (2012). MapReduce Workload Modeling with Statistical Approach. *Journal of Grid Computing, 10*(2), 279-310. doi: 10.1007/s10723-011-9201-4

Yang, X.-S. (2010). *Engineering optimization: an introduction with metaheuristic applications*. Wiley.

Yang, Z., Meratnia, N., & Havinga, P. (2010). Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *IEEE Communications Surveys & Tutorials, 12*(2), 159-170. doi: 10.1109/surv.2010.021510.00088

Yanhua, Y., Meina, S., Zhijun, R., & Junde, S. (2011). *Network traffic analysis and prediction based on APM* présentée à 2011 6th International Conference on Pervasive Computing and Applications. doi: 10.1109/icpca.2011.6106517

Youssef, A., Delpha, C., & Diallo, D. (2016). An optimal fault detection threshold for early detection using Kullback–Leibler Divergence for unknown distribution data. *Signal Processing, 120*, 266-279. doi: 10.1016/j.sigpro.2015.09.008

Zhang-Jian, D.-J., Lee, C.-N., & Hwang, R.-H. (2014). An energy-saving algorithm for cloud resource management using a Kalman filter. *International Journal of Communication Systems, 27*(12), 4078-4091. doi: 10.1002/dac.2599

Zhou, X., Li, R., Chen, T., & Zhang, H. (2016). Network slicing as a service: enabling enterprises' own software-defined cellular networks. *IEEE Communications Magazine, 54*(7), 146-153. doi: 10.1109/mcom.2016.7509393